Software Description

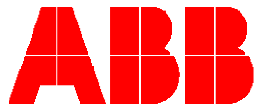
AC500

Scalable PLC
for Individual Automation

AC500
Function Block Libraries

AC500 Libraries

Error Messages of the Function Block Libraries
External System Library
Internal System Library
Modbus Library
Series90 AC500 Library
ASCII Communication Library
CS31 Library
Ethernet Library
ARCNET Library
PROFIBUS Library
CANopen Library
DeviceNet Library
DC541 Library
Counter Library
Diagnosis Library



Contents

Error Messages of the Function Block Libraries 2

0000hex...0FFFhex - Telegram error 2

1000hex...1FFFhex - Device error 3

2000hex...2FFFhex - Interface error 4

3000hex...3FFFhex - Protocol error 5

4000hex...4FFFhex - Block input error 6

5000hex...5FFFhex - Request error 6

6000hex...6FFFhex – Coupler error 7

Error messages of the Function Block Libraries

0000hex...0FFFhex - Telegram error

DEC	HEX	Error description
0	0000	No error
1	0001	COM_MOD_MAST: Error message from slave ILLEGAL FUNCTION ETH_MOD_MAST:Error message from slave ILLEGAL FUNCTION
2	0002	COM_MOD_MAST: Error message from slave ILLEGAL DATA ADDRESS ETH_MOD_MAST: Error message from slave ILLEGAL DATA ADDRESS
3	0003	COM_MOD_MAST: Error message from slave ILLEGAL DATA VALUE ETH_MOD_MAST: Error message from slave ILLEGAL DATA VALUE
4	0004	COM_MOD_MAST: Error message from slave SLAVE DEVICE FAILURE ETH_MOD_MAST: Error message from slave SLAVE DEVICE FAILURE
5	0005	COM_MOD_MAST: Error message from slave ACKNOWLEDGE ETH_MOD_MAST: Error message from slave ACKNOWLEDGE
6	0006	COM_MOD_MAST: Error message from slave SLAVE DEVICE BUSY ETH_MOD_MAST: Error message from slave SLAVE DEVICE BUSY
8	0008	COM_MOD_MAST: Error message from slave MEMORY PARITY ERROR ETH_MOD_MAST: Error message from slave MEMORY PARITY ERROR
10	000A	COM_MOD_MAST: Error message from slave GATEWAY PATH UNAVAILABLE ETH_MOD_MAST: Error message from slave GATEWAY PATH UNAVAILABLE
11	000B	COM_MOD_MAST: Error message from slave GATEWAY TARGET DEVICE FAILED TO RESPOND ETH_MOD_MAST: Error message from slave GATEWAY TARGET DEVICE FAILED TO RESPOND

1000hex...1FFFhex - Device error

DEC	HEX	Error description
4097	1001	Device does not exist
4098	1002	Command not supported by the device. The function is not supported by the device firmware/hardware. Block library newer than the device firmware. FC..: No high-speed counter available at the given module.
4100	1004	Error operating mode. FC..: Operating mode "0" -> No counter set in the PLC configuration.
4101	1005	Invalid status FLASH_READ: Block is not written yet FLASH_WRITE: Block was already written RETAIN..: No program loaded
4117	1015	Format error SD..: File cannot be read because of an invalid format. Data could not be read or not be read completely.
4119	1017	Incorrect length PERSISTENT..: Data have an incorrect length RETAIN..: Data have an incorrect length
4120	1018	Checksum error
4123	101B	Device access error Flash..: Resources are not available PERSISTENT..: Data could not be copied, access error or no data do exist RETAIN..: Data could not be copied, access error or no data do exist SD..: Access to the SD card is not possible (e.g. memory exhausted, file already opened, etc.
4124	101C	Incorrect number PERSISTENT..: Because of the current CPU parameters, data are loaded only partly
4127	101F	Access protection SD..: SD card is write protected
4128	1020	Error when opening SD..: Error when opening a file stored on the SD card
4129	1021	Not found SD..: The searched sector could not be found in the file TASK_INFO: Unknown task
4130	1022	End reached SD..: Section end or end of file reached PERSISTENT..: Because of the current CPU parameters, data are not loaded
4131	1023	Reading error FLASH..: Reading error in data segment: Incorrect checksum
4132	1024	Writing error FLASH..: Block cannot be programmed SD..: File could not be deleted or written
8191	1FFF	Not ready Flash..: The command is already executed by an other instance SD..: Command cannot be executed. Another instance is already active.

2000hex...2FFFhex - Interface error

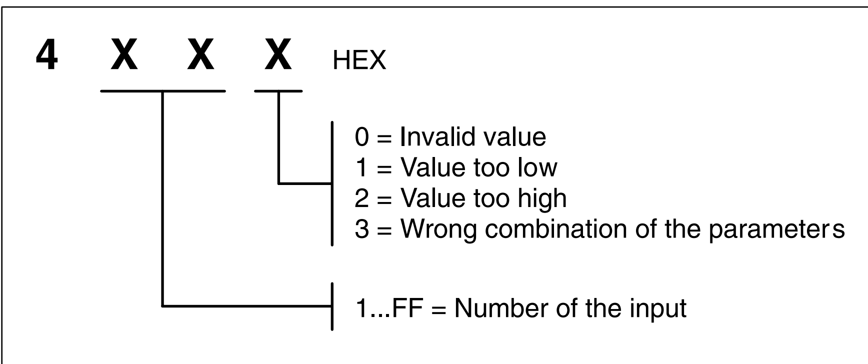
DEC	HEX	Error description
8193	2001	Invalid interface or coupler number COM..: Interface is not configured in the “free mode”
8194	2002	Command not supported by the interface. The function is not supported by the device firmware. Block library newer than the device firmware.
8195	2003	Invalid interface or coupler type. Block is not suitable for this type.
8211	2013	Timeout COM_MOD_MAST: Slave did not respond within the specified time
8212	2014	Framing error (incorrect baud rate, number of stop bits and/or bits per character)
8213	2015	Parity error
8214	2016	Idle error COM..: Character timeout occurred
8215	2017	Invalid length COM_MOD_MAST: Invalid data length received COM_REC: Received more data than expected
8216	2018	Checksum error
8217	2019	Handshake error
8218	201A	Service failed COM_REC: Unknown error message of the interface COM_SET_PROT: Interface hardware not accessible. Initialization already failed during system start-up.
8219	201B	Access error IO..: Module number does not exist
8220	201C	Incorrect number (quantity) COM_SET_PROT: Invalid protocol index. Index is not supported by the device firmware. IO..: Invalid module number
8223	201F	Access denied COM..: Access to the interface is not possible at present. Control Builder, SYCON.net, OPC or another program is logged in via the interface.

3000hex...3FFFhex - Protocol error

DEC	HEX	Error description
12289	3001	Unknown protocol or protocol not configured.
12290	3002	Command not supported by the protocol. The function is not supported by the device firmware. Block library newer than the device firmware.
12291	3003	Another protocol is configured. FC_DC551: The selected interface (COMx) is not set to the CS31 protocol.
12292	3004	Operating mode error COM_MOD_MAST: Invalid operating mode (master/slave).
12293	3005	Protocol status error Fieldbus coupler .._SYS_DIAG: Master is not in the OPERATE state.
12311	3017	Incorrect length ARC..: Buffer is full CAN2..: Total length of all messages too high ETH_UDP..: Buffer is full.
12314	301A	Execution failed COM_SET_PROT: Initialization of the protocol failed
12315	301B	Access error ARC..: Buffer does not exist / is not specified CAN2..: Buffer does not exist / is not specified ETH_UDP..: Buffer does not exist / is not specified IO..: There is no module in the selected slot
12316	301C	Wrong number IO..: Invalid module number > max.
12319	301F	Access denied COM..: Access to the interface is not possible at present. Control Builder, SYCON.net, OPC or another program is logged in via the interface.
12320	3020	Error when opening ARC..: Error during protocol initialization. Protocol not yet ready. CAN2..: Error during protocol initialization. Protocol not yet ready. ETH_UDP..: Error during protocol initialization. Protocol not yet ready.
12325	3025	Address error COM_MOD_MAST: Receive telegram does not contain the expected register address.
12326	3026	Function error COM_MOD_MAST: The received FCT does not correspond to the sent FCT.
12327	3027	Invalid value COM_MOD_MAST: Receive telegram contains an unexpected value.
16383	3FFF	Not ready. Resources currently not available. COM_MOD_MAST: Transmission is not possible at the moment. Another instance of the block is already transmitting. COM_SEND: Transmission is not possible at the moment. Another instance of the block is already transmitting.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:



5000hex...5FFFhex - Request error5000hex...5FFFhex - Request error

DEC	HEX	Error description
20482	5002	The request is not supported (e.g. in the simulation mode of the Control Builder).
20503	5017	Incorrect length ARC..: Invalid data length CAN2..: Invalid DLC in message ETH_UDP..: Invalid data length
20507	501B	Access error COM_MOD_MAST: Invalid memory address DATA or DATA + NB. At least one datum is outside of the access range of the user program. The range includes different flag ranges. ETH_MOD_MAST: Invalid memory address DATA or DATA + NB. At least one datum is outside of the access range of the user program. The range includes different flag ranges.
20508	501C	Invalid number (quantity) CAN2A_SEND: Invalid number of messages at input NUM COM_MOD_MAST: Invalid number of data at NB (0 or more than permitted). ETH_MOD_MAST: Invalid number of data at NB (0 or more than permitted).
20517	5025	Address error ARC..: Invalid IP address CAN2..: Invalid identifier in message COM_MOD_MAST: Invalid slave address. Broadcast not permitted in connection with the selected function code. ETH_MOD_MAST: Invalid slave address. Broadcast not permitted in connection with the selected function code. ETH_UDP..: Invalid IP address.
20518	5026	Function error COM_MOD_MAST: Invalid function code FCT. ETH_MOD_MAST: Invalid function code FCT.

6000hex...6FFFhex - Coupler error

DEC	HEX	Error description
24578	6002	CAN..: Service was rejected by the node with SDO abortion. Index/subindex not valid or no access to the specified node. DNM..: Resource not available or invalid class ID DPM../DPV1..: Resource not available. Free buffer memory in slave is not sufficient for the requested service.
24579	6003	DPM../DPV1..: Requested service (e.g. DPV1) is not active in the slave.
24584	6008	DNM..: Service not available in module. Read/write function not supported by the selected class.
24585	6009	DNM..: Attribute invalid or not supported DPM../DPV1..: No data received from slave
24587	600B	DNM..: Request is already in progress
24588	600C	DNM..: Conflict of the object status
24590	600E	DNM..: Attribute cannot be set or writing is not permitted
24591	600F	DNM..: Permission check faulty or access denied
24592	6010	DNM..: Status conflict. Device prohibits execution
24593	6011	CAN..: No response from the selected node DNM..: No response from the selected device DPM../DPV1..: No response received from slave.
24594	6012	DPM../DPV1..: Master not in logical "token ring".
24595	6013	CAN..: Selected node is not ready for operation DNM..: Not enough receiving data
24596	6014	CAN..: Local resources are not available. Requested bus parameters are not available. Coupler is not configured. DNM..: Local resources are not available. Requested bus parameters are not available. Coupler is not configured.
24597	6015	CAN..: Parameter error DNM..: Parameter error
24598	6016	DNM..: Object does not exist
24599	6017	Received data length too big. Internal buffer too small.
24601	6019	DPM../DPV1..: Unexpected reaction of slave or reaction not in accordance with standard.
24624	6030	CAN..: Function timeout DNM..: Device not configured
24626	6032	DNM..: Format error in the received data ETH_UDP..: TCP/UDP task not available or IP task not ready.
24627	6033	CAN..: Maximum buffer size of the receiving data exceeded ETH_UDP..: Internal task with configuration data not available
24628	6034	CAN..: Function not available. Code unknown. DNM..: Code unknown ETH_MOD..: Invalid parameter for "ServerConnection" ETH_UDP..: No MAC address available
24629	6035	CAN..: Unknown area. Buffer exceeded. DNM..: Overflow of buffer length ETH_MOD..: Invalid parameter for "Task Timeout" ETH_UDP..: Waiting for warm start performed by the application
24630	6036	CAN..: Unknown function in HOST message or function still active DNM..: Other service still active ETH_MOD..: Invalid parameter for "OBM Timeout" ETH_UDP..: Unknown flag in start parameters DPM../DPV1..: Slave denied access to the requested data
24631	6037	CAN..: Parameter error DNM..: Parameter error or MAC ID beyond the valid range

		ETH_MOD..: Invalid parameter for "Mode" ETH_UDP..: Invalid IP address in start parameters
24632	6038	ETH_MOD..: Invalid parameter for "Send Timeout" ETH_UDP..: Invalid subnet mask in start parameters
24633	6039	CAN..: Sequence error DNM..: Sequence error or one MAC ID was multiple used in one network ETH_MOD..: Invalid parameter for "Connect Timeout" ETH_UDP..: Invalid gateway IP in start parameters
24634	603A	ETH_MOD..: Invalid parameter for "Close Timeout".
24635	603B	CAN..: Data error DNM..: Data error ETH_MOD..: Invalid parameter for "Swab" ETH_UDP..: Unknown device type
24636	603C	CAN..: Node address configured twice DNM..: Display of total number of data sets incorrect ETH_MOD..: TCP task not ready ETH_UDP..: Access to IP address in the specified source failed
24637	603D	CAN..: ADD table incorrect DNM..: ADD table incorrect ETH_MOD..: PLC task not ready ETH_UDP..: Initialization of the driver layer failed
24638	603E	CAN..: Total length of the node parameters incorrect DNM..: Size of the I/O configuration table incorrect ETH_MOD..: Error during initialization ETH_UDP..: No source specified for IP address (BOOTP, DHCP, IP address parameter)
24639	603F	CAN..: Transmission type unknown DNM..: I/O configuration does not match with the ADD table
24640	6040	CAN..: Length of the PDO-cfg file too big DNM..: Parameter size incorrct or channel/handler already in use
24641	6041	CAN..: Unknown baud rate DNM..: Number of defined inputs in the ADD table does not match with the I/O configuration
24642	6042	CAN..: COB-ID SYNC beyond the valid range DNM..: Number of defined outputs in the ADD table does not match with the I/O
24643	6043	CAN..: Value of the synchronization timer invalid DNM..: Unknown data type in the I/O configuration
24644	6044	CAN..: Input offset of the PDOs too big DNM..: Defined data type of an I/O module does not match with the defined data size
24645	6045	CAN..: Output offset of the PDOs too big DNM..: The configured output address of an I/O module is not within the permitted address range of 3584 bytes
24646	6046	CAN..: Inconsistency between the PDO and the ADD table DNM..: The configured input address of an I/O module is not within the permitted address range of 3584 bytes
24647	6047	CAN..: Length of the ADD table inconsistent DNM..: Unknown connection type
24648	6048	CAN..: Total data length inconsistent DNM..: Several identical connections defined
24649	6049	CAN..: COB-ID Emergency beyond the permitted range DNM..: The configured value of the "Exp_Packet_Rate" of a connection is smaller than the value of the "Prod_Inhibit_Time"
24650	604A	CAN..: COM-ID Node Guard beyond the permitted range DNM..: Inconsistent parameter field "DNM_PRED_MSTSL_CFG_DATA"
24651	604B	CAN..: Configured PDO length greater than 8 DNM..: Device could not perform "Duplicate_MAC-ID check". Incorrect baud rate or

		no connection to the device possible.
24652	604C	CAN..: Number of defined objects in SDO data too big DNM..: Value of "usRecFragTimer" beyond the permitted range
24686	606E	ETH_UDP..: Timeout has occurred
24687	606F	ETH_MOD..: Unknown send or receive telegram ETH_UDP..: Invalid timeout parameter
24688	6070	ETH_MOD..: TCP responds with an error ETH_UDP..: Invalid socket
24689	6071	ETH_MOD..: No corresponding socket found ETH_UDP..: Command cannot be executed in the current socket state
24690	6072	ETH_MOD..: Command with invalid value
24691	6073	ETH_MOD..: TCP task status error ETH_UDP..: No access to target IP address
24692	6074	ETH_UDP..: Invalid option parameter
24693	6075	ETH_MOD..: No free socket found ETH_UDP..: Invalid command parameter
24694	6076	ETH_MOD..: TCP command is directed to an unknown socket ETH_UDP..: Invalid IP address or no access to address
24695	6077	ETH_MOD..: Time for a client job is over ETH_UDP..: Invalid port number or port not available
24696	6078	ETH_MOD..: Socket has been closed unexpectedly ETH_UDP..: Connection closed
24697	6079	ETH_MOD..: Not-Ready flag has been set by the user ETH_UDP..: Connection reset
24698	607A	ETH_MOD..: OMB task cannot open socket ETH_UDP..: Invalid protocol
24699	607B	ETH_MOD..: Watchdog event in PLC task, only in I/O mode ETH_UDP..: No socket available
24700	607C	ETH_MOD..: TCP task in configuration state ETH_UDP..: Invalid MAC address
24701	607D	ETH_MOD..: PLC task not initialized
24702	607E	ETH_MOD..: Server socket was closed without response from the device
24705	6081	DPM../DPV1..: DPV1 not in "OPEN" state
24706	6082	ETH_UDP..: Invalid mode parameter DPM../DPV1..: Invalid parameters received from slave. Communication stopped.
24707	6083	ETH_UDP..: Maximum data length exceeded or ARP cache full DPM../DPV1..: Service still active. Parallel operation is not possible
24708	6084	ETH_UDP..: Maximum number of messages exceeded DPM../DPV1..: Data length too high for the reserved buffer
24709	6085	ETH_UDP..: Maximum number of IP multicast groups exceeded DPM../DPV1..: Wrong parameter
24710	6086	ETH_UDP..: ARP input not found in ARP cache
24725	6095	ETH_UDP..: Invalid message response received
24727	6097	ETH_MOD..: Invalid message length ETH_UDP..: Invalid message length
24728	6098	CAN..: Unknown message command DNM..: Unknown message command ETH_MOD..: Unknown message command ETH_UDP..: Unknown message command
24730	609A	DPM../DPV1..: Invalid message command
24732	609C	ETH_UDP..: Sequence error during transmission in Sequence Message Mode
24734	609E	ETH_UDP..: Command cannot be executed or command is currently executed
24736	60A0	ETH_MOD..: Error in telegram header

24737	60A1	CAN..: Node address beyond the permitted range DNM..: Device address beyond the permitted range ETH_MOD..: Invalid address detected in the telegram DPM../DPV1..: Invalid slave address
24738	60A2	CAN..: Invalid address range DNM..: Invalid address range
24739	60A3	CAN..: Data buffer overflow DNM..: Data buffer overflow ETH_MOD..: Invalid data address
24741	60A5	CAN..: Incorrect data counter DNM..: Incorrect data counter ETH_MOD..: Invalid data counter
24742	60A6	CAN..: Unknown data type DNM..: Unknown data type
24743	60A7	CAN..: Unknown function DNM..: Unknown function ETH_MOD..: OBM task received an error in the response of the TCP task
24776	60C8	CAN..: Coupler is not configured DNM..: Coupler is not configured ETH_UDP..: Task not initialized
24778	60CA	ETH_MOD..: OBM task does not have a segment from RCS
24779	60CB	ETH_MOD..: Unknown or invalid sender specified with the command
24786	60D2	ETH_UDP..: No configuration data available
24788	60D4	ETH_UDP..: Error while reading the configuration data
24789	60D5	ETH_UDP..: Error while creating the diagnosis structure
24794	60DA	ETH_UDP..: Not enough memory available

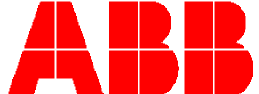
Software Description

AC500

Scalable PLC
for Individual Automation

External System
Function Block Library

External
System
Library



Contents

External System Library 2

Preconditions for the use of the library 2

Special characteristics of the external system library 2

Components of the external system library 2

Overview of blocks arranged according to their call names 2

BATT Reading the battery status 3

CLOCK Display and set clock 4

CLOCK_DT Display and set clock in „DT“ format 8

Glossary 10

Index 12

External System Library

Preconditions for the use of the library

Note:

The blocks contained in the external system library can only be executed in RUN mode of the PLC, but not in simulation mode.

Special characteristics of the external system library

Components of the external system library

The external system library contains the following function blocks:

Group: Battery	
BATT	Reading the battery status

Group: Data access	
CPUCheckUserAccess	Reserved for internal processing
CPUDevItfCmd	Reserved for internal processing

Group: Real-time clock	
CLOCK	Setting/reading the clock and date
CLOCK_DT	Setting/reading the clock and date in "DT" format

Overview of blocks arranged according to their call names

Character description:

FBhV … Function block with historical values

FBnohV … Function block without historical values

F … Function

VE name	Type	Function
BATT	F	Reading the battery status
CLOCK	FBhv	Setting/reading the clock and date
CLOCK_DT	FBhv	Setting/reading the clock and date in "DT" format
CPUCheckUserAcess	F	Reserved for internal processing
CPUDevItfCmd	F	Reserved for internal processing

BATT Reading the battery status

```
            BATT
BATT_EN--EN         ----BATT_LOAD
```

The block BATT reads the charge state of the battery.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysExt_AC500_V10.LIB	

Block type

Function

Parameter

EN	Input	BOOL	Enabling of the block processing
	Output	BYTE	Charge state of the battery

Description

Using the block BATT the charge state of the battery can be requested.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the charge state of the battery is available at the block output.

(Output) STRING(16)

The output of the block BATT outputs the charge state of the battery. The following values are possible at present:

0: Battery empty

1: Remaining battery charge below 20 %

2: Battery charge OK

Function call in IL

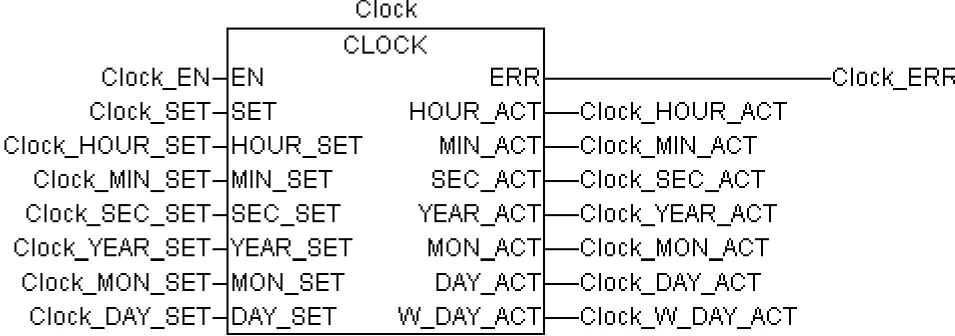
```
LD    BATT_EN
BATT
ST    BATT_LOAD
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
BATT_LOAD := BATT(BATT_EN);
```

CLOCK Display and set clock



This function block allows to set and display (read) the current time and date.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysExt_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		CLOCK	Instance name
EN	Input	BOOL	Enabling of the block processing
SET	Input	BOOL	Setting the time and date to the values available at the set inputs.
HOUR_SET	Input	BYTE	Set input for hours
MIN_SET	Input	BYTE	Set input for minutes
SEC_SET	Input	BYTE	Set input for seconds
YEAR_SET	Input	WORD	Set input for the year
MON_SET	Input	BYTE	Set input for the month
DAY_SET	Input	BYTE	Set input for the day
ERR	Output	BOOL	Error message of the block
HOUR_ACT	Output	BYTE	Hour
MIN_ACT	Output	BYTE	Minute
SEC_ACT	Output	BYTE	Second
YEAR_ACT	Output	WORD	Year
MON_ACT	Output	BYTE	Month
DAY_YCT	Output	BYTE	Day
W_DAY_ACT	Output	BYTE	Day of week (number)

Description

This function block allows to set and display the current time and date. The clock is set by means of the set inputs for the time and date. The values available at the set inputs are read in with the occurrence of a FALSE/TRUE edge at input SET. As long as a TRUE signal is present at the EN input, the current date and time are displayed at the block outputs.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current time and date are available at the outputs.

SET BOOL (set)

With an occurring FALSE/TRUE edge at input SET, the clock is set to the values available at the time and date inputs.

HOUR_SET BYTE (hour set)

Set input for the hours. The clock works in 24 hours mode, i.e. it changes from 23:59:59 h to 0:0:0 h.

Valid value range: 0...23.

MIN_SET BYTE (minute set)

Set input for the minutes.

Valid value range: 0...59.

SEC_SET BYTE (second set)

Set input for the seconds.

Valid value range: 0...59.

YEAR_SET WORD (year set)

Set input for the year. Four-digit input, e.g. 2005.

MON_SET BYTE (month set)

Set input for the month.

Valid value range: 1...12.

DAY_SET BYTE (day set)

Set input for the day (which day of the month).

Valid value range: 1...31.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing.

HOUR_ACT BYTE (hour actual)

Current value for the hours.

MIN_ACT BYTE (minute actual)

Current value for the minutes.

SEC_ACT BYTE (second actual)

Current value for the seconds.

YEAR_ACT WORD (year actual)

Current value for the year.

MON_ACT BYTE (month actual)

Current value for the month.

DAY_ACT BYTE (day actual)

Current value for the day.

W_DAY_ACT BYTE (week day actual)

Current value for the number of the day of week.

Function call in IL

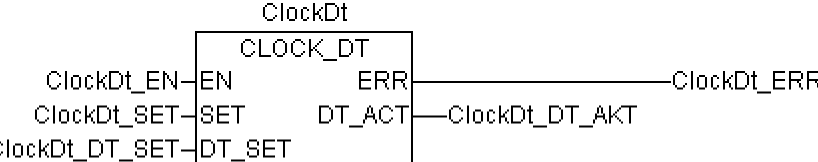
```
CAL  Clock(
     EN := Clock_EN,
     SET := Clock_SET,
     HOUR_SET := Clock_HOUR_SET,
     MIN_SET := Clock_MIN_SET,
     SEC_SET := Clock_SEC_SET,
     YEAR_SET := Clock_YEAR_SET,
     MON_SET := Clock_MON_SET,
     DAY_SET := Clock_DAY_SET)

LD   Clock.ERR
ST   Clock_ERR

LD   Clock.HOUR_ACT
ST   Clock_HOUR_ACT

LD   Clock.MIN_ACT
ST   Clock_MIN_ACT

LD   Clock.SEC_ACT
ST   Clock_SEC_ACT

LD   Clock.YEAR_ACT
ST   Clock_YEAR_ACT

LD   Clock.MON_ACT
ST   Clock_MON_ACT

LD   Clock.DAY_ACT
ST   Clock_DAY_ACT

LD   Clock.W_DAY_ACT
ST   Clock_W_DAY_ACT
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
Clock  (EN := Clock_EN,
        SET := Clock_SET,
        HOUR_SET := Clock_HOUR_SET,
        MIN_SET := Clock_MIN_SET,
        SEC_SET := Clock_SEC_SET,
        YEAR_SET := Clock_YEAR_SET,
        MON_SET := Clock_MON_SET,
        DAY_SET := Clock_DAY_SET);

Clock_ERR := Clock.ERR;
Clock_HOUR_ACT := Clock.HOUR_ACT;
Clock_MIN_ACT := Clock.MIN_ACT;
Clock_SEC_ACT := Clock.SEC_ACT;
Clock_YEAR_ACT := Clock.YEAR_ACT;
Clock_MON_ACT := Clock.MON_ACT;
Clock_DAY_ACT := Clock.DAY_ACT;
Clock_W_DAY_ACT := Clock.W_DAY_ACT;
```

CLOCK_DT Display and set clock in "DT" format



This function block allows to set and display (read) the current time and date in the standardized "DT" format.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysExt_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		CLOCK_DT	Instance name
EN	Input	BOOL	Enabling of the block processing
SET	Input	BOOL	Setting the time and date to the value available at the set input "DT_SET".
DT_SET	Input	DT	Set input for time and date in "DT" format
ERR	Output	BOOL	Error message of the block
DT_ACT	Output	DT	Current time and date in "DT" format

Description

This function block allows to set and display the current time and date in the standardized "DT" format. Setting of the clock is performed via set input "DT_SET". The value available at the set input is read in with the occurrence of a FALSE/TRUE edge at input SET. As long as a TRUE signal is present at the EN input, the current time and date are displayed in the standardized "DT" format at block output "DT_ACT".

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current time and date are available at output "DT_ACT".

SET BOOL (set)

With the occurrence of a FALSE/TRUE edge at input "SET", the time and date are set to the value available at input "DT_SET".

DT_SET BYTE (date and time set)

Set input for time and date in standardized "DT" format. The input for a "DT" value always has to start with the preceding designation "DT#", followed by the date, a hyphen and the time.

Valid value range: DT#1970-01-01-00:00:00 to DT#2106-02-06-06:28:15

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing.

DT_ACT BYTE (date and time actual)

Current value for date and time in standardized "DT" format.

Function call in IL

```
CAL   ClockDt(
      EN := ClockDt_EN,
      SET := ClockDt_SET,
      DT_SET := ClockDt_DT_SET)

LD    ClockDt.ERR
ST    ClockDt_ERR

LD    ClockDt.DT_ACT
ST    ClockDt_DT_ACT
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
Clock   (EN := ClockDt_EN,
        SET := ClockDt_SET,
        DT_SET := ClockDt_DT_SET);

ClockDt_ERR := ClockDt.ERR;
ClockDt_DT_ACT := ClockDt.DT_ACT;
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

Components of the external system library 2

E

External System Library

BATT Reading the battery status 3

CLOCK Setting/reading the clock and date 4

CLOCK_DT Setting/reading the clock and date in "DT" format 8

O

Overview of blocks arranged according to their call names 2

S

Special characteristics of the external system library 2

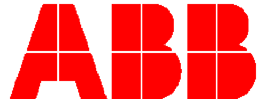
Software Description

AC500

Scalable PLC
for Individual Automation

Internal System
Function Block Library

Internal
System
Library



Contents

Internal System Library 3

Preconditions for the use of the library 3

Special characteristics of the internal system library 3

Components of the internal system library 3

Overview of blocks arranged according to their call names 5

COM_SET_PROT Setting/changing COM protocol actively 6

CPU_INFO Reading the CPU type 9

DIAG_ACK Acknowledging an error 11

DIAG_ACK_ALL Acknowledging all errors of an error class 14

DIAG_EVENT Generating an error event 16

DIAG_GET Reading an error 19

DIAG_INFO Displaying an overview of all errors not yet read 24

FLASH_DEL Deleting a data segment in Flash memory 26

FLASH_READ Reading a data set from Flash memory 29

FLASH_WRITE Writing a data set to Flash memory 33

FPU_EXCEPTION_INFO Reading information about the FPU exception 37

IO_DIAG Reading the diagnosis data of the I/O-Bus 40

IO_INFO Reading the number of devices connected to the I/O-Bus 43

IO_MODULE_DIAG Reading the module diagnosis data of the I/O-Bus 45

IO_VERSION Reading the version of the I/O-Bus driver 48

PERSISTENT_CLEAR Delete persistent data from SRAM 50

PERSISTENT_EXPORT Write persistent data from RAM-DISC to SD Card 52

PERSISTENT_IMPORT Write persistent data from SD Card to RAM-DISC 54

PERSISTENT_RESTORE Write persistent data from RAM-DISC to SRAM 56

PERSISTENT_SAVE Write persistent data from SRAM to RAM-DISC 58

RETAIN_CLEAR Delete retain data from SRAM 60

RETAIN_EXPORT Write retain data from RAM-DISC to SD Card 62

RETAIN_IMPORT Write retain data from SD Card to RAM-DISC 64

RETAIN_RESTORE Write retain data from RAM-DISC to SRAM 66

RETAIN_SAVE Write retain data from SRAM to RAM-DISC 68

RTS_INFO Reading the version of the CPU runtime system 70

Structure of the file USRDATXX.DAT stored on the SD card 72
SD_READ Reading a data segment from the SD card 74
SD_WRITE Writing a data segment to the SD card 80
SLOT_INFO Reading the slot information 86
SYS_TIME Reading the system time 89
Glossary 90
Index 92

Internal System Library

Preconditions for the use of the library

Note:

The blocks contained in the internal system library can only be executed in RUN mode of the PLC, but not in simulation mode.

Special characteristics of the internal system library

The internal system library contains all generally applicable blocks for the system, i.e. blocks for general system diagnosis functions or system information. The only exception is the "Data Storage" subgroup which contains special FLASH and SD card blocks. Using these blocks, data can be either stored to the Flash memory or to the SD card, as desired.

When creating a new project, the library SysInt_AC500_V10.lib is automatically included to the Control Builder project.

Components of the internal system library

The internal system library contains the following function blocks:

Group: Data Storage		**Page**
Subgroup: Flash		
FLASH_DEL	Deleting a data segment in the Flash memory	26
FLASH_READ	Reading a data set from the Flash memory	29
FLASH_WRITE	Writing a data set to the Flash memory	33
Subgroup: Persistent data		
PERSISTENT_CLEAR	Delete persistent data from SRAM	50
PERSISTENT_EXPORT	Write persistent data from RAM-DISC to SD Card	52
PERSISTENT_IMPORT	Write persistent data from SD Card to RAM-DISC	54
PERSISTENT_RESTORE	Write persistent data from RAM-DISC to SRAM	56
PERSISTENT_SAVE	Write persistent data from SRAM to RAM-DISK	58
Subgroup: Retain data		
RETAIN_CLEAR	Delete retain data from SRAM	60
RETAIN_EXPORT	Write retain data from RAM-DISC to SD Card	62
RETAIN_IMPORT	Write retain data from SD Card to RAM-DISC	64
RETAIN_RESTORE	Write retain data from RAM-DISC to SRAM	66
RETAIN_SAVE	Write retain data from from SRAM to RAM-DISK	68
Subgroup: SD Card		
SD_READ	Reading a data segment from the SD card	74
SD_WRITE	Writing a data segment to the SD card	80

Group: Diagnosis		**Page**
DIAG_ACK	Acknowledging an error	11
DIAG_ACK_ALL	Acknowledging all errors of an error class	14
DIAG_EVENT	Generating an error event	16
DIAG_GET	Reading an error	19
DIAG_INFO	Displaying an overview of all errors not yet read	24

Group: I/O bus		Page
IO_DIAG	Reading the diagnosis data of the I/O bus	40
IO_INFO	Reading the number of devices connected to the I/O bus	43
IO_MODULE_DIAG	Reading the module diagnosis data of the I/O bus	45
IO_VERSION	Reading the version of the I/O bus driver	48

Group: Serial interface		Page
COM_SET_PROT	Setting/changing COM protocol actively	6

Group: System information		Page
CPU_INFO	Reading the CPU type	9
FPU_EXCEPTION_INFO	Reading information about the FPU exception	37
RTS_INFO	Reading the version of the CPU runtime system	70
SLOT_INFO	Reading the slot information	86
SYS_TIME	Reading the system time	89

Overview of blocks arranged according to their call names

Character description:

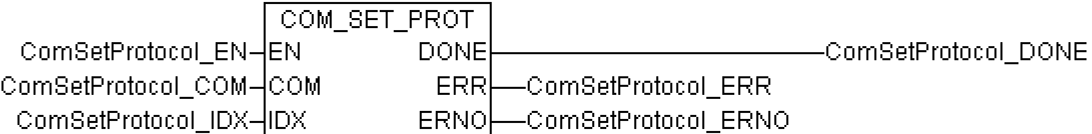
FBhV … Function block with historical values

FBnohV … Function block without historical values

F … Function

VE name	Type	Function	Page
COM_SET_PROT	FBnohv	Setting/changing COM protocol actively	6
CPU_INFO	FBnohv	Reading the CPU type	9
DIAG_ACK	FBhv	Acknowledging an error	11
DIAG_ACK_ALL	FBhv	Acknowledging all errors of an error class	14
DIAG_EVENT	FBhv	Generating an error event	16
DIAG_GET	FBhv	Reading an error	19
DIAG_INFO	FBhv	Displaying an overview of all errors not yet read	24
FLASH_DEL	FBhv	Deleting a data segment in the Flash memory	26
FLASH_READ	FBhv	Reading a data set from the Flash memory	29
FLASH_WRITE	FBhv	Writing a data set to the Flash memory	33
FPU_EXCEPTION_INFO	FBhv	Reading information about the FPU exception	37
IO_DIAG	FBnohv	Reading the diagnosis data of the I/O bus	40
IO_INFO	FBnohv	Reading the number of devices connected to the I/O bus	43
IO_MODULE_DIAG	FBnohv	Reading the module diagnosis data of the I/O bus	45
IO_VERSION	FBnohv	Reading the version of the I/O bus driver	48
PERSISTENT_CLEAR	FBnohv	Delete persistent data from SRAM	50
PERSISTENT_EXPORT	FBnohv	Write persistent data from RAM-DISC to SD Card	52
PERSISTENT_IMPORT	FBnohv	Write persistent data from SD Card to RAM-DISC	54
PERSISTENT_RESTORE	FBnohv	Write persistent data from RAM-DISC to SRAM	56
PERSISTENT_SAVE	FBnohv	Write persistent data from SRAM to RAM-DISC	58
RETAIN_CLEAR	FBnohv	Delete retain data from SRAM	60
RETAIN_EXPORT	FBnohv	Write retain data from from RAM-DISC to SD Card	62
RETAIN_IMPORT	FBnohv	Write retain data from SD Card to RAM-DISC	64
RETAIN_RESTORE	FBnohv	Write retain data from RAM-DISC to SRAM	66
RETAIN_SAVE	FBnohv	Write retain data from SRAM to RAM-DISC	68
RTS_INFO	FBnohv	Reading the version of the CPU runtime system	70
SD_READ	FBhv	Reading a data segment from the SD card	74
SD_WRITE	FBhv	Writing a data segment to the SD card	80
SLOT_INFO	FBnohv	Reading the slot information	86
SYS_TIME	F	Reading the system time	89

COM_SET_PROT Setting/changing COM protocol actively



With the function block COM_SET_PROT, the serial interfaces of the CPU can actively be set on a predefined protocol or changed between several protocols from the user program.

Block data

Available as of PLC runtime system:	V2.0	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values

Parameters

EN	Input	BOOL	Enabling of the block processing
COM	Input	BYTE	Interface identification (COM1 or COM2)
IDX	Input	BYTE	Index number
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

With the setting "COMx – Multi[SLOT]" in the PLC configuration of the serial interfaces, several protocols can be predefined per COM. Using the function block COM_SET_PROT, the user can switch between these protocols from the user program. The rising edge (FALSE->TRUE) at the input EN activates the protocol which was selected over the inputs COM and IDX.

If the function block is used with a serial interface on which only one protocol was defined, the interface is initialized newly and the protocol started again.

Other applications of the function block can be e.g. a change of the baud rate or also a RESET defined of a protocol.

EN BOOL (enable)

With a FALSE->TRUE edge at the input EN, the processing of the function block is activated. If the values are valid at the inputs, the function block is processed. If the inputs are not valid, the processing is stopped and a corresponding error appears at the outputs ERR/ERNO. The function block processing is always finished when the value at output DONE changes to TRUE. During the processing, signal changes at the inputs are noticed but not evaluated.

COM BYTE (communication)

At input COM, the number of the serial input is defined.

COM = 1: COM1
COM = 2: COM2

IDX BYTE (index)

The index number of the protocol, which is to be activated on the serial interface COM, is indicated at the input IDX. The assignment of the index number to the individual protocols is carried out in the PLC configuration of the serial interfaces. If several protocols are defined in the setting "COMx – Multi[SLOT]", the index number IDX = 0 corresponds to the protocol in the first place, and IDX = 1 corresponds to the protocol in the second place. In all other cases, if only one protocol is defined per interface, the index number IDX = 0 must be used.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

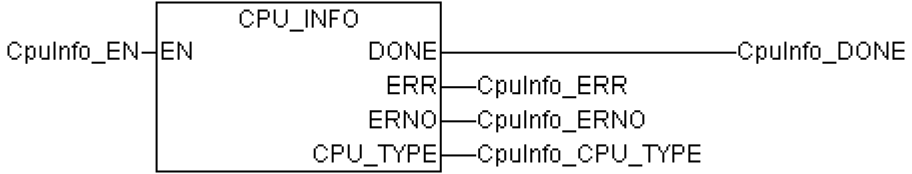
```
CAL  COM_SET_PROT(
     EN := ComSetProtocol_EN,
     COM := ComSetProtocol_COM,
     IDX := ComSetProtocol_IDX)

LD   COM_SET_PROT.DONE
ST   ComSetProtocol_DONE

LD   COM_SET_PROT.ERR
ST   ComSetProtocol_ERR

LD   COM_SET_PROT.ERNO
ST   ComSetProtocol_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
COM_SET_PROT (EN  := ComSetProtocol_EN,
              COM := ComSetProtocol_COM,
              IDX := ComSetProtocol_IDX);

ComSetProtocol_DONE := COM_SET_PROT.DONE;
ComSetProtocol_ERR  := COM_SET_PROT.ERR;
ComSetProtocol_ERNO := COM_SET_PROT.ERNO;
```

CPU_INFO Reading the CPU type



The block CPU_INFO reads the CPU type.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values

Parameter

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
CPU_TYPE	Output	BYTE	CPU type

Description

Using the block CPU_INFO the CPU type can be read.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

CPU_TYPE BYTE (CPU type)

CPU_TYPE outputs the type of the CPU.

The following values are possible:

```
Value CPU type

20    PM571
21    PM581
22    PM591
23    PM582
```

Function call in IL

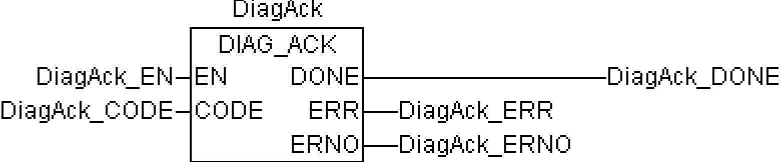
```
CAL   CPU_INFO(
      EN := CpuInfo_EN)

LD    CPU_INFO.DONE
ST    CpuInfo_DONE

LD    CPU_INFO.ERR
ST    CpuInfo_ERR

LD    CPU_INFO.ERNO
ST    CpuInfo_ERNO

LD    CPU_INFO.CPU_TYPE
ST    CpuInfo_CPU_TYPE
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
CPU_INFO(EN := CpuInfo_EN);

CpuInfo_DONE := CPU_INFO.DONE;
CpuInfo_ERR := CPU_INFO.ERR;
CpuInfo_ERNO := CPU_INFO.ERNO;
CpuInfo_CPU_TYPE := CPU_INFO.CPU_TYPE;
```

DIAG_ACK Acknowledging an error



The block DIAG_ACK can be used to acknowledge any error.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		DIAG_ACK	Instance name
EN	Input	BOOL	Enabling of the block processing
CODE	Input	DWORD	Code number of the error to be acknowledged
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block DIAG_ACK can be used to acknowledge any error. Selection of the error to be acknowledged is performed using a 32 bit code. If the error list contains several entries with the selected error code number, acknowledgement is always performed for the oldest entry in the list.

EN BOOL (enable)

Acknowledgement is activated with a FALSE -> TRUE edge at input EN. The acknowledge request is processed, if the value applied at input CODE is valid. If the value at input CODE is not valid, processing is aborted and a corresponding error is displayed at output ERR/ERNO. If output DONE changes to TRUE, the acknowledgement is completed successfully. During processing of the request, state changes at input EN are recognized but not evaluated.

CODE DWORD (code)

At input CODE, the code number of the error to be acknowledged is specified. The code number of an existing error can be read using the block DIAG_GET or calculated manually. The structure of the error coding is as follows:

	31 30	29 28	27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
32 bits –>	0 0	x x	x x x x	x x x x x x x x	x x x x x	x x x x x	x x x x x x
		CLASS	COMPONENT	DEVICE	MODULE	CHANNEL	ERROR

Bit 0 to 5	- Error number	Valid range: 0...63
Bit 6 to 10	- Channel number	Valid range: 0...31
Bit 11 to 15	- Module number	Valid range: 0...31
Bit 16 to 23	- Device number	Valid range: 0...255
Bit 24 to 27	- Component number	Valid range: 0...15
Bit 28 to 29	- Error class	Valid range: 1...4
Bit 30 to 31	- Reserved; both bits always must be zero.	

Valid range: 16#3E9..16#3FFFFFFF

DONE BOOL (done)

Output DONE indicates the processing state of the acknowledgement. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the acknowledgement request. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

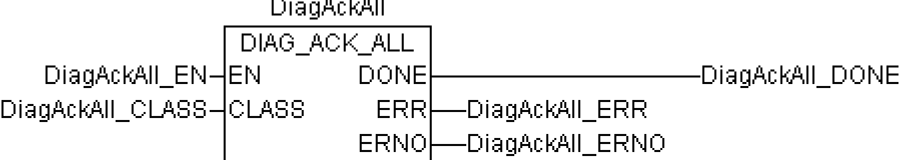
```
CAL  DiagAck(
     EN := DiagAck_EN
     CODE := DiagAck_CODE)

LD   DiagAck.DONE
ST   DiagAck_DONE

LD   DiagAck.ERR
ST   DiagAck_ERR

LD   DiagAck.ERNO
ST   DiagAck_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DiagAck(EN := DiagAck_EN,
       CODE := DiagAck_CODE);

DiagAck_DONE := DiagAck.DONE;
DiagAck_ERR  := DiagAck.ERR;
DiagAck_ERNO := DiagAck.ERNO;
```

DIAG_ACK_ALL Acknowledging all errors of an error class



The block DIAG_ACK_ALL can be used to acknowledge all errors of an error class.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		DIAG_ACK_ALL	Instance name
EN	Input	BOOL	Enabling of the block processing
CLASS	Input	BYTE	Error class to be acknowledged
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block DIAG_ACK_ALL can be used to acknowledge all errors of an error class simultaneously. Selection of the error class, the errors of which are to be acknowledged, is done using the input CLASS.

EN BOOL (enable)

Acknowledgement of all errors of the error class specified at input CLASS is activated with a FALSE -> TRUE edge at input EN. The acknowledge request is processed, if the value at input CLASS is valid. If the value at input CLASS is not valid, processing is aborted and a corresponding error is displayed at output ERR/ERNO. If output DONE changes to TRUE, the acknowledgement is completed successfully. During processing of the request, state changes at input EN are recognized but not evaluated.

CLASS BYTE (class)

At input CLASS, the error class is specified, the errors of which are to be acknowledged.

Valid range: 1...4

DONE BOOL (done)

Output DONE indicates the processing state of the acknowledgement. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the acknowledgement request. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

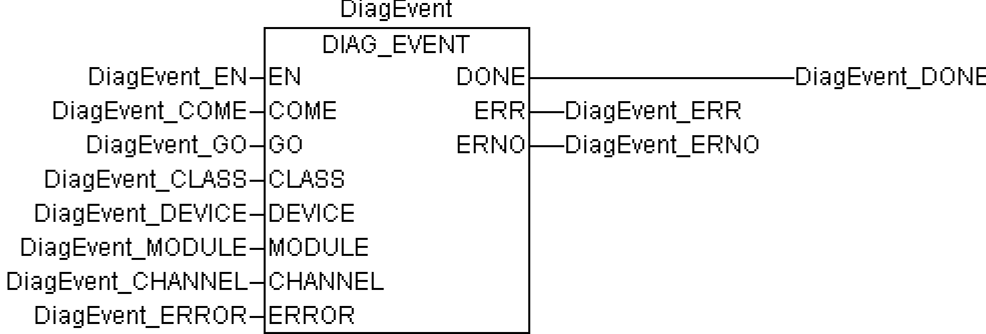
```
CAL  DiagAckAll(
     EN := DiagAckAll_EN
     CODE := DiagAckAll_CLASS)

LD   DiagAckAll.DONE
ST   DiagAckAll_DONE

LD   DiagAckAll.ERR
ST   DiagAckAll_ERR

LD   DiagAckAll.ERNO
ST   DiagAckAll_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DiagAckAll(EN := DiagAckAll_EN,
        CLASS := DiagAckAll_CLASS);

DiagAckAll_DONE := DiagAckAll.DONE;
DiagAckAll_ERR  := DiagAckAll.ERR;
DiagAckAll_ERNO := DiagAckAll.ERNO;
```

DIAG_EVENT Generating an error event



The block DIAG_EVENT generates a programmable error event.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		DIAG_EVENT	Instance name
EN	Input	BOOL	Enabling of the block processing
COME	Input	BOOL	Error status "COME"
GO	Input	BOOL	Error status "GO"
CLASS	Input	BYTE	Error class to be acknowledged
DEVICE	Input	BYTE	Device number
MODULE	Input	BYTE	Module number
CHANNEL	Input	BYTE	Channel number
ERROR	Input	BYTE	Error number
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block DIAG_EVENT can be used to generate any error of component 15 (User error). Each error can have three possible states: 1) "Error has come" (COME), 2)"Error has gone" (GO) and 3) "Error has been acknowledged". Using the block DIAG_EVENT, errors of the states 1) and 2) can be generated. Error acknowledgement is done using the blocks DIAG_ACK or DIAG_ACK_ALL or using the PLC Browser or directly using the display.

EN BOOL (enable)

Block processing is activated with a FALSE -> TRUE edge at input EN. The block's job is processed, if the values applied at the inputs are valid. Otherwise, if the input values are not valid, processing is aborted and a corresponding error is displayed at output ERR/ERNO. Block processing is completed, if output DONE changes to TRUE. During job processing, state changes at input EN are recognized but not evaluated.

COME BOOL (come)

Input COME is used to specify the status of the error to be generated. This input can be used in combination with input GO.

COME = TRUE = "Error has come"

GO BOOL (go)

Input GO is used to specify the status of the error to be generated. This input can be used in combination with input COME.

GO = TRUE = "Error has gone"

CLASS BYTE (class)

At input "CLASS", the error class to be generated is specified.

Valid range: 1...4

DEVICE BYTE (device)

Input "DEVICE" is used to specify the device number the generated error should be assigned to.

Valid range: 0...255

MODULE BYTE (module)

Input "MODULE" is used to specify the module the generated error should be assigned to.

Valid range: 0...31

CHANNEL BYTE (channel)

Input "CHANNEL" is used to specify the channel the generated error should be assigned to.

Valid range: 0...31

ERROR BYTE (error)

At input "ERROR", the error number to be generated is specified. In order to facilitate later assignment of the errors, it is recommended to use the error numbers already defined. Please refer to the corresponding chapter for a more detailed overview of possible system errors and the used error numbers. If an error is not yet declared by an error number, it is recommended to use a free error number.

Valid range: 0...63

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

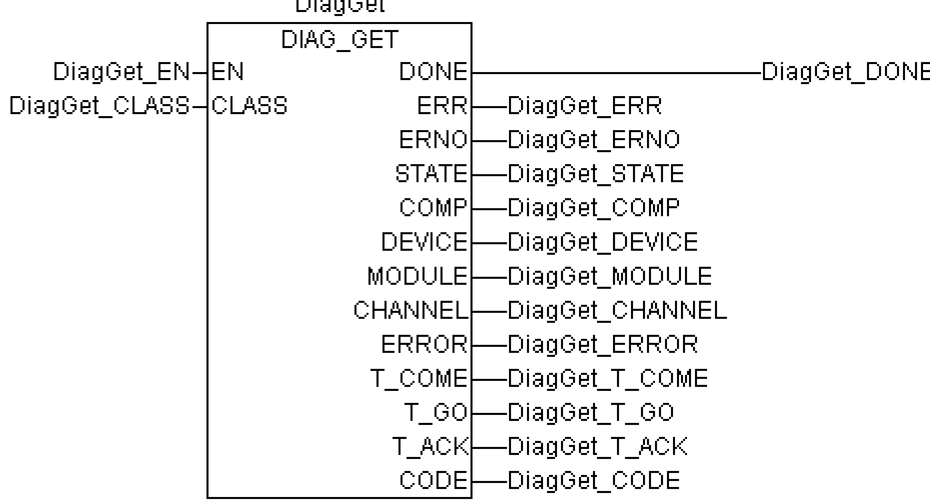
```
CAL  DiagEvent(
     EN := DiagEvent_EN,
     COME := DiagEvent_COME,
     GO := DiagEvent_GO,
     CLASS := DiagEvent_CLASS,
     DEVICE := DiagEvent_DEVICE,
     MODULE := DiagEvent_MODULE,
     CHANNEL := DiagEvent_CHANNEL,
     ERROR := DiagEvent_ERROR)

LD   DiagEvent.DONE
ST   DiagEvent_DONE

LD   DiagEvent.ERR
ST   DiagEvent_ERR

LD   DiagEvent.ERNO
ST   DiagEvent_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DiagEvent(EN := DiagEvent_EN,
        COME := DiagEvent_COME,
        GO := DiagEvent_GO,
        CLASS := DiagEvent_CLASS,
        DEVICE := DiagEvent_DEVICE,
        MODULE := DiagEvent_MODULE,
        CHANNEL := DiagEvent_CHANNEL,
        ERROR := DiagEvent_ERROR);

DiagEvent_DONE := DiagEvent.DONE;
DiagEvent_ERR  := DiagEvent.ERR;
DiagEvent_ERNO := DiagEvent.ERNO;
```

DIAG_GET Reading an error



The block "DIAG_GET" reads the oldest unread error of any error class.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		DIAG_GET	Instance name
EN	Input	BOOL	Enabling of the block processing
CLASS	Input	BYTE	Error class
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
STATE	Output	BYTE	Error status
COMP	Output	BYTE	Component number
DEVICE	Output	BYTE	Device number
MODULE	Output	BYTE	Module number
CHANNEL	Output	BYTE	Channel number
ERROR	Output	BYTE	Error number
T_COME	Output	DT	Time stamp, if an error has come
T_GO	Output	DT	Time stamp, if an error has gone
T_ACK	Output	DT	Time stamp, if an error has been acknowledged
CODE	Output	DWORD	Code number of the error

Description

The block "DIAG_GET" can be used to read an error of any error class. Each error can only be read once. If this block is used more than once for a specific error class, the next error output is the oldest error that has not been read yet. If all errors were already read or if there is no existing error available, this is accordingly indicated at output STATE.

EN BOOL (enable)

Block processing is activated with a FALSE -> TRUE edge at input EN. The block is processed, if the value at input CLASS is valid. Otherwise, if the input value is not valid, processing is aborted and a corresponding error is displayed at output ERR/ERNO. Block processing is completed, if output DONE changes to TRUE. During job processing, state changes at input EN are recognized but not evaluated.

CLASS BYTE (class)

At input "CLASS", the error class to be read is specified.

Valid range: 1...4

DEVICE BYTE (device)

Input "DEVICE" is used to specify the device number the generated error should be assigned to.

Valid range: 0...255

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

STATE BYTE (state)

STATE outputs the current status of the read error. The error status is a combination of the states "Error has come", "Error has gone" and "Error has been acknowledged". The following status numbers are possible:

State	Error	come	gone	acknowledged
16#02		x		
16#04			x	
16#06		x	x	
16#08				x
16#0A		x		x
16#0C			x	x
16#0E		x	x	x
16#F0	Alle errors read or no error available			

COMP BYTE (component)

"COMP" outputs the component number the read error is assigned to.

DEVICE BYTE (device)

"DEVICE" outputs the device number the read error is assigned to.

MODULE BYTE (module)

"MODULE" outputs the module the read error is assigned to.

CHANNEL BYTE (channel)

"CHANNEL" outputs the channel the read error is assigned to.

ERROR BYTE (error)

"ERROR" outputs the error number of the read error.

T_COME DT (time come)

T_COME outputs the time stamp, when the read error occurred ("has come"). If no time stamp is available for the error status "come" (see also the description of output STATE), no value is written to this output. In this case, the output value remains at the default value DT#1970-01-01-00:00.

T_GO DT (time go)

T_GO outputs the time stamp, when the read error "has gone". If no time stamp is available for the error status "gone" (see also the description of output STATE), no value is written to this output. In this case, the output value remains at the default value DT#1970-01-01-00:00.

T_ACK DT (time acknowledge)

T_ACK outputs the time stamp, when the read error was "acknowledged". If no time stamp is available for the error status "acknowledged" (see also the description of output STATE), no value is written to this output. In this case, the output value remains at the default value DT#1970-01-01-00:00.

CODE   DWORD   (code)

CODE outputs the code number of the read error. The structure of the error encoding is as follows:

	31 30	29 28	27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
32 bits –>	0 0	x x	x x x x	x x x x x x x x	x x x x x	x x x x x	x x x x x x
		CLASS	COMPONENT	DEVICE	MODULE	CHANNEL	ERROR

Function call in IL

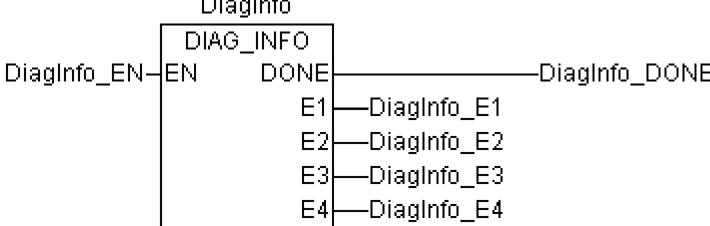
```
CAL  DiagGet(
     EN := DiagGet_EN,
     CLASS := DiagGet_CLASS)

LD   DiagGet.DONE
ST   DiagGet_DONE

LD   DiagGet.ERR
ST   DiagGet_ERR

LD   DiagGet.ERNO
ST   DiagGet_ERNO

LD   DiagGet.STATE
ST   DiagGet_STATE

LD   DiagGet.COMP
ST   DiagGet_COMP

LD   DiagGet.DEVICE
ST   DiagGet_DEVICE

LD   DiagGet.MODULE
ST   DiagGet_MODULE

LD   DiagGet.CHANNEL
ST   DiagGet_CHANNEL

LD   DiagGet.ERROR
ST   DiagGet_ERROR

LD   DiagGet.T_COME
ST   DiagGet_T_COME

LD   DiagGet.T_GO
ST   DiagGet_T_GO

LD   DiagGet.T_ACK
ST   DiagGet_T_ACK

LD   DiagGet.CODE
ST   DiagGet_CODE
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DiagGet(EN := DiagGet_EN,
        CLASS := DiagGet_CLASS);

DiagGet_DONE := DiagGet.DONE;
DiagGet_ERR := DiagGet.ERR;
DiagGet_ERNO := DiagGet.ERNO;
DiagGet_STATE := DiagGet.STATE;
DiagGet_COMP := DiagGet.COMP;
DiagGet_DEVICE := DiagGet.DEVICE;
DiagGet_MODULE := DiagGet.MODULE;
DiagGet_CHANNEL := DiagGet.CHANNEL;
DiagGet_ERROR := DiagGet.ERROR;
DiagGet_T_COME := DiagGet.T_COME;
DiagGet_T_GO := DiagGet.T_GO;
DiagGet_T_ACK := DiagGet.T_ACK;
DiagGet_CODE := DiagGet.CODE;
```

DIAG_INFO Displaying an overview of all errors not yet read



The block DIAG_INFO displays an overview of all errors that were not read yet.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		DIAG_INFO	Instance name
EN	Input	BOO	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
E1	Output	BOOL	At least 1 unread error is present in error class 1.
E2	Output	BOOL	At least 1 unread error is present in error class 2.
E3	Output	BOOL	At least 1 unread error is present in error class 3.
E4	Output	BOOL	At least 1 unread error is present in error class 4.

Description

The block DIAG_INFO can be used to display an overview of all errors that were not read yet. The output is sorted according to the error classes E1 to E4. If at least one error is present in any error class and if this error has not been read yet using the block DIAG_GET, this is displayed at the corresponding Ex output.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. If the processing is finished, DONE is set to TRUE.

E1 BOOL (error class 1)

Output E1 is set to TRUE, if at least 1 error is present in error class 1 and if this error has not been read yet using the block DIAG_GET. Consequently, if E1 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

E2 BOOL (error class 2)

Output E2 is set to TRUE, if at least 1 error is present in error class 2 and if this error has not been read yet using the block DIAG_GET. Consequently, if E2 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

E3 BOOL (error class 3)

Output E3 is set to TRUE, if at least 1 error is present in error class 3 and if this error has not been read yet using the block DIAG_GET. Consequently, if E3 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

E4 BOOL (error class 4)

Output E4 is set to TRUE, if at least 1 error is present in error class 4 and if this error has not been read yet using the block DIAG_GET. Consequently, if E4 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

Function call in IL

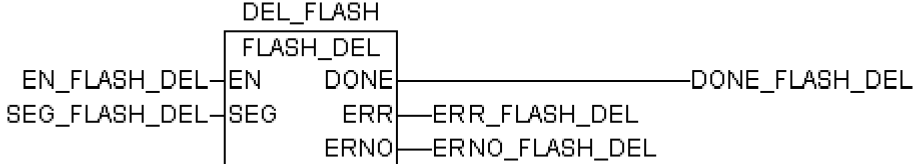
```
CAL  DiagInfo(
     EN := DiagGInfo_EN)

LD   DiagInfo.DONE
ST   DiagInfo_DONE

LD   DiagInfo.E1
ST   DiagInfo_E1

LD   DiagInfo.E2
ST   DiagInfo_E2

LD   DiagInfo.E3
ST   DiagInfo_E3

LD   DiagInfo.E4
ST   DiagInfo_E4
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DiagInfo(EN := DiagInfo_EN);

DiagInfo_DONE := DiagInfo.DONE;
DiagInfo_E1 := DiagInfo.E1;
DiagInfo_E2 := DiagInfo.E2;
DiagInfo_E3 := DiagInfo.E3;
DiagInfo_E4 := DiagInfo.E4;
```

FLASH_DEL Deleting a data segment in Flash memory



This function block deletes a user data segment from the Flash. All data in this data segment are lost after deletion.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block with historical values

Parameters

Instance		FLASH_DEL	Instance name
EN	Input	BOOL	Deletion of the data segment by a FALSE/TRUE edge
SEG	Input	BYTE	Number of the data segment; 1 or 2
DONE	Output	BOOL	Deletion procedure completed
ERR	Output	BOOL	Error occurred, data segment cannot be deleted
ERNO	Output	WORD	Error number

Description

This function block deletes a user data segment from the Flash. All data in this data segment are lost after deletion.

Important note:

Access to the Flash is only possible using the function blocks FLASH_DEL, FLASH_WRITE and FLASH_READ.

The input SEG defines the data segment within the Flash. In the AC500, two segments with the number 1 and 2 are reserved for the user, each providing 64 kbytes. Deleting a data segment within the Flash may take several PLC cycles.

A FALSE/TRUE edge at input EN triggers the deletion of the data segment once. Input EN will not be evaluated again, until the delete operation is completed (DONE = TRUE).

After completion of the delete procedure, all function block outputs are updated. The deletion was successful, if DONE = TRUE and ERR = FALSE. If the outputs show DONE = TRUE and ERR = TRUE, the data segment could not be deleted.

EN BOOL (enable)

Processing of the block is controlled via input EN.

EN = FALSE/TRUE edge:

Deletion of the data segment is started once. Input EN will not be evaluated again, until the delete operation is completed (DONE = TRUE).

EN = TRUE:

The block is not processed, i. e. it no longer changes its outputs. This is not valid during a delete operation.

SEG BYTE (segment)

At input SEG, the number of the data segment in the Flash is specified. In the AC500, controller two data segments are available for the user.

Valid values: 1 and 2 respectively

DONE BOOL (done)

Output DONE indicates that the data segment deletion is completed. This output has always to be considered together with output ERR.

The following applies:

DONE = TRUE and ERR = FALSE:

The deletion procedure is completed. The data segment has been deleted successfully.

DONE = TRUE and ERR = TRUE:

An error occurred while deleting the data segment. The data segment could not be deleted.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the delete operation. This output always has to be considered together with output DONE. If the data segment could not be deleted, the following applies: DONE = TRUE and ERR = TRUE. Output ERNO signalizes the error number.

ERNO WORD (error number)

The output ERNO indicates an error number. This output has always to be considered together with the outputs DONE and ERR.

The functions FLASH_DEL, FLASH_WRITE and FLASH_READ are executed in the background by the operating system. This procedures can take quite long time, since the PLC user program is processed with priority. Output ERNO then indicates that the block execution is finished (0x0FFF = BUSY).

During this phase, the outputs ERR and DONE are set to FALSE.

The inputs and outputs can neither be duplicated nor inverted.

Function call in IL

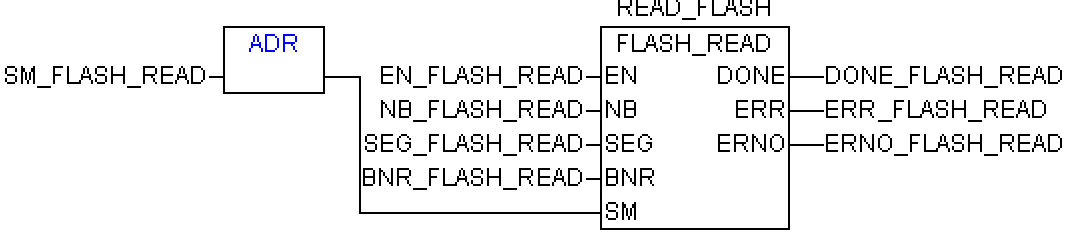
```
CAL  DEL_FLASH(EN:=EN_FLASH_DEL,
     SEG := SEG_FLASH_DEL)

LD   DEL_FLASH.DONE
ST   DONE_FLASH_DEL

LD   DEL_FLASH.ERR
ST   ERR_FLASH_DEL

LD   DEL_FLASH.ERNO
ST   ERNO_FLASH_DEL
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
DEL_FLASH(EN := EN_FLASH_DEL,
SEG := SEG_FLASH_DEL);
DONE_FLASH_DEL := DEL_FLASH.DONE;
ERR_FLASH_DEL := DEL_FLASH.ERR;
ERNO_FLASH_DEL := DEL_FLASH.ERNO;
```

FLASH_READ Reading a data set from Flash memory



The function block reads a data set from a data segment of the Flash and stores the read data set beginning at the start flag defined by SM.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block with historical values

Parameters

Instance		FLASH_READ	Instance name
EN	Input	BOOL	Storage of a data set by a FALSE/TRUE edge
NB	Input	WORD	Number of blocks of the data set; 1...1927
SEG	Input	BYTE	Number of the data segment; 1 or 2
BNR	Input	WORD	Number of the block within the data segment: 0...1926
SM	Input	DWORD	Address of the start flag of the data set
DONE	Output	BOOL	Read procedure completed
ERR	Output	BOOL	Error has occurred
ERNO	Output	WORD	Error number

Description

The function block reads a data set from a data segment of the Flash and stores the read data set beginning at the start flag defined by SM. The data of the data set have been previously stored to the Flash using the function block FLASH_WRITE.

Important note:

Access to the Flash is only possible using the function blocks FLASH_WRITE and FLASH_READ.

NB blocks are read starting at block BNR within segment SEG and stored starting at address SM.

Either 32 binary data or 16 word data or 8 double word data are read per block.

One block contains 34 bytes:
32 bytes of data
1 byte CRC checksum
1 byte "written identifier"

(see figure at the end of this block description)

Reading a data set is triggered once by a FALSE/TRUE edge at input EN. If no error occurred when reading the data, DONE is set to TRUE and the outputs ERR and ERNO are set to FALSE. The data set is stored beginning at the defined start flag SM. Storing the data set can take several PLC cycles.

If an error occurs during reading, DONE and ERR are both set to TRUE. The error type is indicated at output ERNO.

EN BOOL (enable)

Processing of the block is controlled via input EN.

The following applies:

EN = FALSE/TRUE edge:

The reading procedure of the data set is carried out once.

EN = TRUE:

The block is not processed, i. e. it no longer changes its outputs.

NB WORD (number)

The number of the data set blocks is specified at input NB. Either 32 binary data or 16 word data or 8 double word data are read per block.

Valid values: 1 ... 1927

Example:

- SM = ADR(%MW0.0) and NB = 1: Storing data from %MW0.0 to %MW0.15
  (1 block = 16 word data)

- SM = ADR(%MW0.0) and NB = 2: Storing data from %MW0.0 to %MW0.31
  (2 blocks = 32 word data)

SEG BYTE (segment)

At input SEG, the number of the data segment in the Flash is specified. In the AC500, controller two data segments are available for the user.

Valid values: 1 and 2 respectively

BNR WORD (block number)

The block number in the data segment is specified at input BNR. Valid values: 0...1926

SM DWORD (source memory)

At input SM, the address of the first variable for storing the data set is specified using an ADR operator.

DONE BOOL (done)

Output DONE indicates that the read operation of the data set is completed. This output has always to be considered together with output ERR.

The following applies:

DONE = TRUE and ERR = FALSE: The read operation is completed. The data set is stored beginning at the defined input SM.

DONE = TRUE and ERR = TRUE: An error occurred while reading the data set. Output ERNO signalizes the error number.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the read operation. This output always has to be considered together with output DONE. If an error has occurred, the following applies: DONE = TRUE and ERR = TRUE. Output ERNO signalizes the error number.

ERNO WORD (error number)

The output ERNO indicates an error number. This output has always to be considered together with the outputs DONE and ERR.

The functions FLASH_DEL, FLASH_WRITE and FLASH_READ are executed in the background by the operating system. This procedures can take quite long time, since the PLC user program is processed with priority. Output ERNO then indicates that the block execution is finished (0x0FFF).

During this phase, the outputs ERR and DONE are set to FALSE.

The inputs and outputs can neither be duplicated nor inverted.

The following figure shows the structure of a Flash segment.

	Byte:	1 | 2	3 | 4	5 | 6	....	29 | 30	31 | 32	33	34
Byte Offset	Block No.	Word 1	Word 2	Word 3	....	Word 15	Word 16	CRC	Written Identifier
0	0								
34	1								
68	2								
...	...								
65450	1925								
65484	1926								

Function call in IL

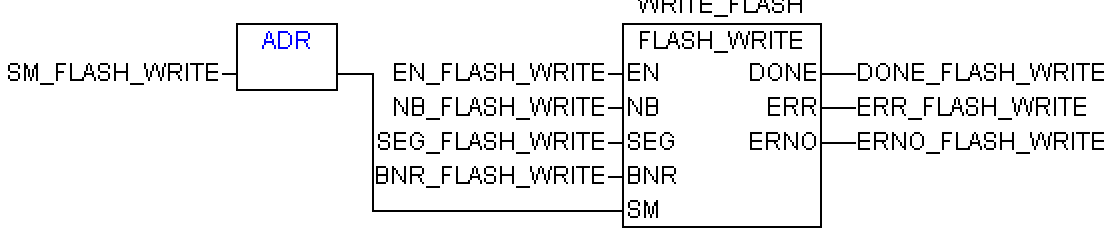
```
CAL  READ_FLASH(EN := EN_FLASH_READ,
     NB := NB_FLASH_READ,
     SEG := SEG_FLASH_READ,
     BNR := BNR_FLASH_READ,
     SM := SM_FLASH_READ)

LD   READ_FLASH.DONE
ST   DONE_FLASH_READ

LD   READ_FLASH.ERR
ST   ERR_FLASH_READ

LD   READ_FLASH.ERNO
ST   ERNO_FLASH_READ
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
READ_FLASH(EN := EN_FLASH_READ,
     NB := NB_FLASH_READ,
     SEG := SEG_FLASH_READ,
     BNR := BNR_FLASH_READ,
     SM := SM_FLASH_READ)
DONE_FLASH_READ := READ_FLASH.DONE;
ERR_FLASH_READ := READ_FLASH.ERR;
ERNO_FLASH_READ := READ_FLASH.ERNO;
```

FLASH_WRITE Writing a data set to Flash memory



The function block writes a data set to a data segment in the Flash.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block with historical values

Parameters

Instance		FLASH_WRITE	Instance name
EN	Input	BOOL	Storage of a data set by a FALSE/TRUE edge
NB	Input	WORD	Number of blocks of the data set; 1...1927
SEG	Input	BYTE	Number of the data segment; 1 or 2
BNR	Input	WORD	Number of the block within the data segment: 0...1926
SM	Input	DWORD	Address of the start flag of the data set
DONE	Output	BOOL	Write operation completed
ERR	Output	BOOL	Error has occurred
ERNO	Output	WORD	Error number

Description

The function block writes a data set to a data segment in the Flash. For that purpose, two data segments are available for the AC500. A delete operation (function block FLASH_DEL) always deletes a complete data segment. A data segment consists of 1927 blocks (0 ... 1926). Each block is composed of 34 bytes.

After a delete operation, data can be written only once to each of these 1927 data segment blocks. If a block containing data is to be overwritten with new data, the entire data segment has to be deleted first. In doing so, all data in this segment are lost.

Access to the Flash is only possible using the function blocks FLASH_DEL, FLASH_WRITE and FLASH_READ.

NB blocks are read starting at address SM and stored in the segment SEG starting at block BNR.

Either 32 binary data or 16 word data or 8 double word data are read per block.

One block contains 34 bytes:
32 bytes of data
1 byte CRC checksum
1 byte "written identifier"
(see figure at the end of this block description)

When a write operation of a data set is started (FALSE/TRUE edge at input EN), the data of the data set must not be changed until the end of the write procedure (DONE = TRUE). Storing the data set in the FLASH can take several PLC cycles.

With a FALSE/TRUE edge at input EN, the data set is written once. Until the storage procedure has not been finished (DONE = TRUE), input EN will not be evaluated again.

After the write operation is completed, the block outputs DONE, ERR and ERNO are updated. The storage was successful, if DONE = TRUE and ERR = FALSE. If DONE = TRUE and ERR = TRUE, an error occurred. The error type is signalized at output ERNO.

A new FALSE/TRUE edge at input EN starts a new write operation. Since without a previous deletion of the data segment no new data can be written to blocks which already contain data, the input BNR must point to the next free block for the next write procedure.

EN BOOL (enable)

Processing of the block is controlled via input EN.

EN = FALSE/TRUE edge:

Writing of the data set is started once. Input EN will not be evaluated again until the storage process is completed (DONE = TRUE).

NB WORD (number)

The number of the data set blocks is specified at input NB. Either 32 binary data or 16 word data or 8 double word data are stored per block.

Valid values: 1...1927

Example:

- SM = ADR(%MW0.0) and NB = 1: Storing the data from %MW0.0 to %MW0.15
  (1 block = 16 word data)

- SM = ADR(%MW0.0) and NB = 2: Storing the data from %MW0.0 to %MW0.31
  (2 blocks = 32 word data)

SEG BYTE (segment)

At input SEG, the number of the data segment in the Flash is specified. In the AC500, controller two data segments are available for the user.

Valid values: 1 and 2 respectively

BNR WORD (block number)

The block number in the data segment is specified at input BNR. Valid values: 0...1926

SM DWORD (source memory)

At input SM, the address of the first variable for storing the data set is specified using an ADR operator. When a write operation of a data set is started (FALSE/TRUE edge at input EN), the data of the data set must not be changed until the end of the write procedure (DONE = TRUE).

DONE BOOL (done)

Output DONE indicates that the write operation of the data set is completed. This output has always to be considered together with output ERR.

The following applies:

DONE = TRUE and ERR = FALSE: The write operation is completed. The data set has been stored in the Flash.

DONE = TRUE and ERR = TRUE: An error occurred during the write operation. Output ERNO signalizes the error number.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the write operation. This output always has to be considered together with output DONE. If an error has occurred, the following applies: DONE = TRUE and ERR = TRUE. Output ERNO signalizes the error number.

ERNO WORD (error number)

The output ERNO indicates an error number. This output has always to be considered together with the outputs DONE and ERR.

The inputs and outputs can neither be duplicated nor inverted.

The following figure shows the structure of a Flash segment.

	Byte:	1 | 2	3 | 4	5 | 6	....	29 | 30	31 | 32	33	34
Byte Offset	Block No.	Word 1	Word 2	Word 3	....	Word 15	Word 16	CRC	Written Identifier
0	0								
34	1								
68	2								
...	...								
65450	1925								
65484	1926								

Function call in IL

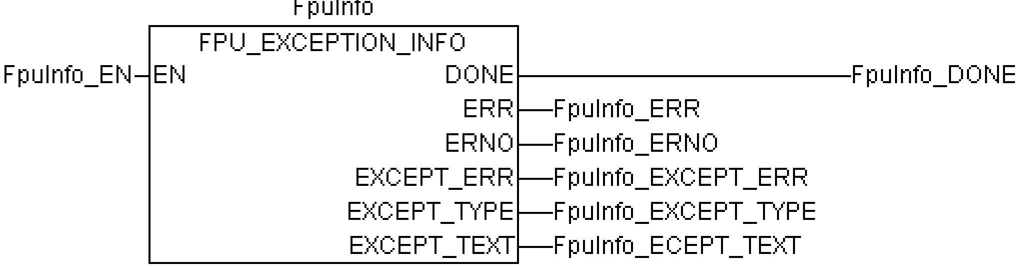
```
CAL   WRITE_FLASH(EN := EN_FLASH_WRITE,
      NB := NB_FLASH_WRITE,
      SEG := SEG_FLASH_WRITE,
      BNR := BNR_FLASH_WRITE,
      SM := SM_FLASH_WRITE)

LD    WRITE_FLASH.DONE
ST    DONE_FLASH_WRITE

LD    WRITE_FLASH.ERR
ST    ERR_FLASH_WRITE

LD    WRITE_FLASH.ERNO
ST    ERNO_FLASH_WRITE
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
WRITE_FLASH(EN := EN_FLASH_WRITE,
      NB := NB_FLASH_WRITE,
      SEG := SEG_FLASH_WRITE,
      BNR := BNR_FLASH_WRITE,
      SM := SM_FLASH_WRITE)
DONE_FLASH_WRITE := WRITE_FLASH.DONE;
ERR_FLASH_WRITE := WRITE_FLASH.ERR;
ERNO_FLASH_WRITE := WRITE_FLASH.ERNO;
```

FPU_EXCEPTION_INFO Reading information about the FPU exception



The function block FPU_EXCEPTION_INFO reads information which has been stored during an FPU exception.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block with historical values

Parameters

Instance		FPU_EXCEPTION_INFO	Instance name
EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Read operation completed
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
EXCEPT_ERR	Output	BOOL	FPU exception occurred
EXCEPT_TYPE	Output	BYTE	FPU exception type
EXCEPT_TEXT	Output	STRING(14)	FPU exception text

Description

The function block FPU_EXCEPTION_INFO reads information which has been stored during an FPU exception. The reading operation takes 1 PLC cycle. If data are available, they are provided by setting the outputs DONE = TRUE and ERR = FALSE. If ausgegeben. If several errors have appeared up to the call of the function block, the last error is shown.

Caution! This function block only has to be used with CPUs which have an FPU. The CPU parameter "Reaction on floating point exception" has to be set to "No failure".

The following table represents an overview of the possible return values of the function block.

EXCEPT_ERR	EXCEPT_TYPE	EXCEPT_TEXT
FALSE	16#00	"No error"
TRUE	16#01	"Zero Divide"
TRUE	16#02	"Overflow"
TRUE	16#03	"Underflow"
TRUE	16#04	"Invalid"
TRUE	16#05	"Inexact"
TRUE	16#06	"Function"

The error "Function" occurs, if a result of a function (e.g. SQRT, LN or ACOS) cannot be calculated.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

EXCEPT_ERR BOOL (exception error)

Output EXCEPT_ERR indicates that an exception error has occurred in the FPU.

EXCEPT_TYPE BYTE (exception type)

Output EXCEPT_TYPE indicates the type of the occurred exception error.

EXCEPT_TEXT STRING(14) (exception text)

Output EXCEPT_TEXT indicates in text form which exception error has occurred.

Function call in IL

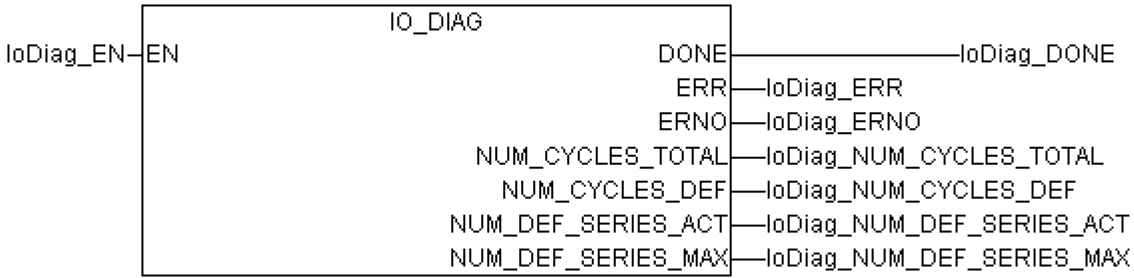
```
CAL  FpuInfo(
     EN := FpuInfo_EN)

LD   FpuInfo.DONE
ST   FpuInfo_DONE

LD   FpuInfo.ERR
ST   FpuInfo_ERR

LD   FpuInfo.ERNO
ST   FpuInfo_ERNO

LD   FpuInfo.EXCEPT_ERR
ST   FpuInfo_EXCEPT_ERR

LD   FpuInfo.EXCEPT_TYPE
ST   FpuInfo_EXCEPT_TYPE

LD   FpuInfo.EXCEPT_TEXT
ST   FpuInfo_EXCEPT_TEXT
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
FpuInfo(EN := FpuInfo_EN);

FpuInfo_DONE := FpuInfo.DONE;
FpuInfo_ERR := FpuInfo.ERR;
FpuInfo_ERNO := FpuInfo.ERNO;
FpuInfo_EXCEPT_ERR := FpuInfo.EXCEPT_ERR;
FpuInfo_EXCEPT_TYPE := FpuInfo.EXCEPT_TYPE;
FpuInfo_EXCEPT_TEXT := FpuInfo.EXCEPT_TEXT;
```

IO_DIAG Reading the diagnosis data of the I/O-Bus



The block IO_DIAG reads the diagnosis data of the I/O bus.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block without historical values

Parameter

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NUM_CYCLES_TOTAL	Output	DWORD	Total number of I/O bus cycles since system start
NUM_CYCLES_DEF	Output	DWORD	Total number of defective I/O bus cycles occurred since system start
NUM_DEF_SERIES_ACT	Output	BYTE	Actual number of successively occurred defective I/O bus cycles
NUM_DEF_SERIES_MAX	Output	BYTE	Maximum number of successively occurred defective I/O bus cycles captured since system start

Description

The block IO_DIAG can be used to read the diagnosis data of the I/O bus.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

NUM_CYCLES_TOTAL DWORD (number of cycles total)

NUM_CYCLES_TOTAL displays the total number of I/O bus cycles performed since system start.

NUM_CYCLES_DEF DWORD (number of cycles defective)

NUM_CYCLES_DEF displays the total number of defective I/O bus cycles occurred since system start.

NUM_DEF_SERIES_ACT BYTE (number of defective cycles in series actual)

NUM_DEF_SERIES_ACT displays the actual number of successively occurred defective I/O bus cycles.

NUM_DEF_SERIES_MAX BYTE (number of defective cycles in series maximal)

NUM_DEF_SERIES_MAX displays the maximum number of successively occurred defective I/O bus cycles captured since system start.

Function call in IL

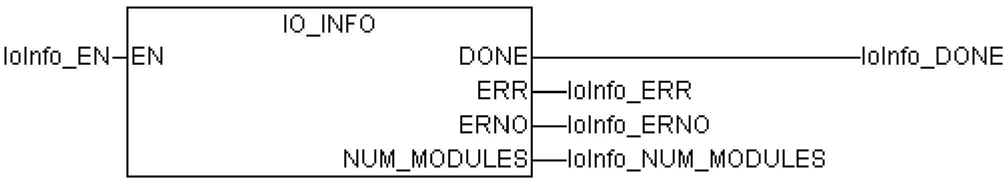
```
CAL  IO_DIAG(
     EN := IoDiag_EN)

LD   IO_DIAG.DONE
ST   IoDiag_DONE

LD   IO_DIAG.ERR
ST   IoDiag_ERR

LD   IO_DIAG.ERNO
ST   IoDiag_ERNO

LD   IO_DIAG.NUM_CYCLES_TOTAL
ST   IoDiag_NUM_CYCLES_TOTAL

LD   IO_DIAG.NUM_CYCLES_DEF
ST   IoDiag_NUM_CYCLES_DEF

LD   IO_DIAG.NUM_DEF_SERIES_ACT
ST   IoDiag_NUM_DEF_SERIES_ACT

LD   IO_DIAG.NUM_DEF_SERIES_MAX
ST   IoDiag_NUM_DEF_SERIES_MAX
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
IO_DIAG (EN := IoDiag_EN);

IoDiag_DONE := IO_DIAG.DONE;
IoDiag_ERR := IO_DIAG.ERR;
IoDiag_ERNO := IO_DIAG.ERNO;
IoDiag_NUM_CYCLES_TOTAL := IO_DIAG.NUM_CYCLES_TOTAL;
IoDiag_NUM_CYCLES_DEF := IO_DIAG.NUM_CYCLES_DEF;
IoDiag_NUM_DEF_SERIES_ACT := IO_DIAG. NUM_DEF_SERIES_ACT;
IoDiag_NUM_DEF_SERIES_MAX := IO_DIAG. NUM_DEF_SERIES_MAX;
```

IO_INFO Reading the number of devices connected to the I/O-Bus



The block IO_INFO displays the number of devices connected to the I/O bus.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block without historical values

Parameter

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NUM_MODULES	Output	BYTE	Number of devices connected to the I/O bus

Description

The block IO_INFO can be used to display the number of devices connected to the I/O bus.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

NUM_MODULES BYTE (number of connected modules)

NUM_MODULES displays the number of devices connected to the I/O bus.

Function call in IL

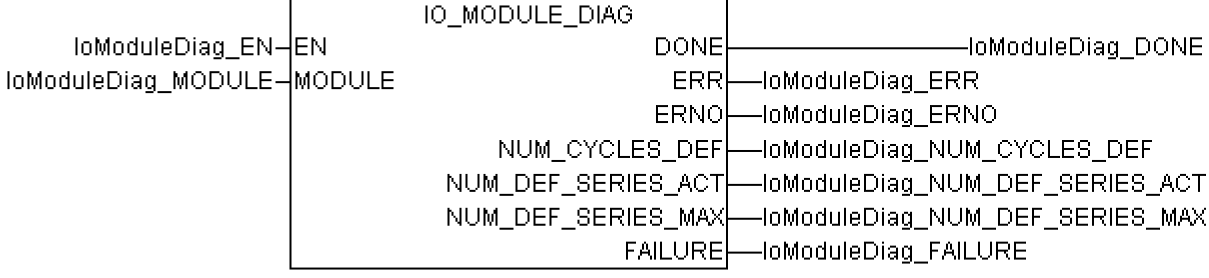
```
CAL   IO_INFO(
      EN := IoInfo_EN)

LD    IO_INFO.DONE
ST    IoInfo_DONE

LD    IO_INFO.ERR
ST    IoInfo_ERR

LD    IO_INFO.ERNO
ST    IoInfo_ERNO

LD    IO_INFO.NUM_MODULES
ST    IoInfo_NUM_MODULES
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
IO_INFO (EN := IoInfo_EN);

IoInfo_DONE := IO_INFO.DONE;
IoInfo_ERR := IO_INFO.ERR;
IoInfo_ERNO := IO_INFO.ERNO;
IoInfo_NUM_ MODULES := IO_INFO.NUM_ MODULES;
```

IO_MODULE_DIAG Reading the module diagnosis data of the I/O-Bus



The block IO_MODULE_DIAG reads the module diagnosis data of the I/O bus.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block without historical values

Parameter

EN	Input	BOOL	Enabling of the block processing
MODULE	Input	BYTE	Module number
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NUM_CYCLES_DEF	Output	DWORD	Total number of defective I/O bus cycles occurred since system start
NUM_DEF_SERIES_ACT	Output	BYTE	Actual number of successively occurred defective I/O bus cycles
NUM_DEF_SERIES_MAX	Output	BYTE	Maximum number of successively occurred defective I/O bus cycles captured since system start
FAILURE	Output	BOOL	Module failure

Description

The block IO_MODULE_DIAG is used to read the module diagnosis data of the I/O bus.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

NUM_CYCLES_DEF DWORD (number of cycles defective)

NUM_CYCLES_DEF displays the total number of defective I/O bus cycles occurred for the selected module since system start.

NUM_DEF_SERIES_ACT BYTE (number of defective cycles in series actual)

NUM_DEF_SERIES_ACT displays the actual number of successive defective I/O bus cycles occurred for the selected module.

NUM_DEF_SERIES_MAX BYTE (number of defective cycles in series maximal)

NUM_DEF_SERIES_MAX displays the maximum number of successive defective I/O bus cycles occurred for the selected module since system start.

FAILURE BOOL (failure)

FAILURE indicates that the maximum allowed number of successively occurring defective I/O bus cycles has been exceeded. In this case the module has to be considered as failed. A failure of one module results in a stop of the entire I/O bus.

Function call in IL

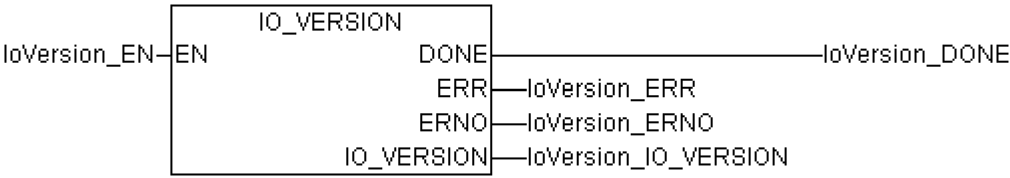
```
CAL  IO_MODULE_DIAG(
     EN := IoModuleDiag_EN,
     MODULE := IoModuleDiag_MODULE)

LD   IO_MODULE_DIAG.DONE
ST   IoModuleDiag_DONE

LD   IO_MODULE_DIAG.ERR
ST   IoModuleDiag_ERR

LD   IO_MODULE_DIAG.ERNO
ST   IoModuleDiag_ERNO

LD   IO_MODULE_DIAG.NUM_CYCLES_DEF
ST   IoModuleDiag_NUM_CYCLES_DEF

LD   IO_MODULE_DIAG.NUM_DEF_SERIES_ACT
ST   IoModuleDiag_NUM_DEF_SERIES_ACT

LD   IO_MODULE_DIAG.NUM_DEF_SERIES_MAX
ST   IoModuleDiag_NUM_DEF_SERIES_MAX

LD   IO_MODULE_DIAG.FAILURE
ST   IoModuleDiag_FAILURE
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
IO_MODULE_DIAG (EN := IoModuleDiag_EN,
               MODULE := IoModuleDiag_MODULE);

IoModuleDiag_DONE := IO_ MODULE_DIAG.DONE;
IoModuleDiag_ERR := IO_ MODULE_DIAG.ERR;
IoModuleDiag_ERNO := IO_ MODULE_DIAG.ERNO;
IoModuleDiag_NUM_CYCLES_DEF := IO_ MODULE_DIAG.NUM_CYCLES_DEF;
IoModuleDiag_NUM_DEF_SERIES_ACT := IO_ MODULE_DIAG. NUM_DEF_SERIES_ACT;
IoModuleDiag_NUM_DEF_SERIES_MAX:= IO_ MODULE_DIAG. NUM_DEF_SERIES_MAX;
IoModuleDiag_FAILURE := IO_ MODULE_DIAG.FAILURE;
```

IO_VERSION Reading the version of the I/O-Bus driver



The block IO_VERSION reads the version of the I/O bus driver.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block without historical values

Parameter

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
IO_VERSION	Output	WORD	Version of the I/O bus driver

Description

The block IO_VERSION reads the version of the I/O bus driver.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

IO_VERSION WORD (IO bus driver version)

IO_VERSION outputs the version of the I/O bus driver.

Example: IO_VERSION = 1000 -> V1.0.0.0

Function call in IL

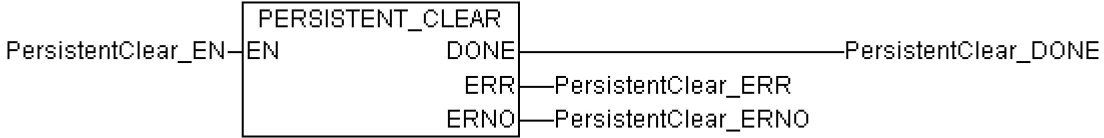
```
CAL  IO_VERSION(
     EN := IoVersion_EN)

LD   IO_VERSION.DONE
ST   IoVersion_DONE

LD   IO_VERSION.ERR
ST   IoVersion_ERR

LD   IO_VERSION.ERNO
ST   IoVersion_ERNO

LD   IO_VERSION.IO_VERSION
ST   IoVersion_IO_VERSION
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
IO_VERSION (EN := IoVersion_EN);

IoVersion_DONE := IO_VERSION.DONE;
IoVersion_ERR := IO_VERSION.ERR;
IoVersion_ERNO := IO_VERSION.ERNO;
IoVersion_IO_VERSION := IO_VERSION.IO_VERSION;
```

PERSISTENT_CLEAR Delete persistent data from SRAM



With the function block PERSISTENT_CLEAR, all written data located in the PERSISTENT area or the %R area can be deleted.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

With a rising edge (False -> True) at input EN of the function block PERSISTENT_CLEAR, all data in the areas PERSISTENT or %R are deleted.

The use of the function block requires that a valid PERSISTENT area or %R area is set in the PLC configuration of the CPU.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

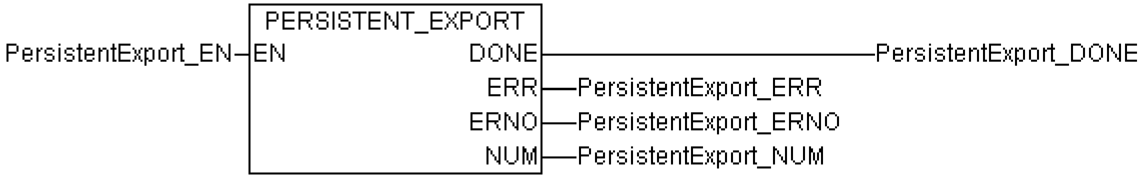
```
CAL  PERSISTENT_CLEAR(
     EN := PersistentClear_EN)

LD   PERSISTENT_CLEAR.DONE
ST   PersistentClear_DONE

LD   PERSISTENT_CLEAR.ERR
ST   PersistentClear_ERR

LD   PERSISTENT_CLEAR.ERNO
ST   PersistentClear_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
PERSISTENT_CLEAR (EN := PersistentClear_EN);

PersistentClear_DONE := PERSISTENT_CLEAR.DONE;
PersistentClear_ERR  := PERSISTENT_CLEAR.ERR;
PersistentClear_ERNO := PERSISTENT_CLEAR.ERNO;
```

PERSISTENT_EXPORT Write persistent data from RAM-DISC to SD Card



With the function block PERSISTENT_EXPORT, all written data located in the PERSISTENT area or the %R area can be written from the RAM-DISC to the SD Card.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	OUTPUT	ARRAY [0..7] OF WORD	Error number of the corresponding segments, stored in an ARRAY [0..7]
NUM	Output	WORD	Number of written segments

Description

With a rising edge (False -> True) at input EN of the function block PERSISTENT_EXPORT, all data in the areas PERSISTENT or %R are written from the RAM-DISC to the SD Card.

The use of the function block requires that a valid PERSISTENT area or %R area is set in the PLC configuration of the CPU.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The ERNO output consists of an ARRAY [0..7], where the appeared error numbers of the segments are stored. The error messages encoding at output ERNO is explained at the beginning of the block description.

NUM WORD (number of areas)

The number of written segments is available at output NUM.

Function call in IL

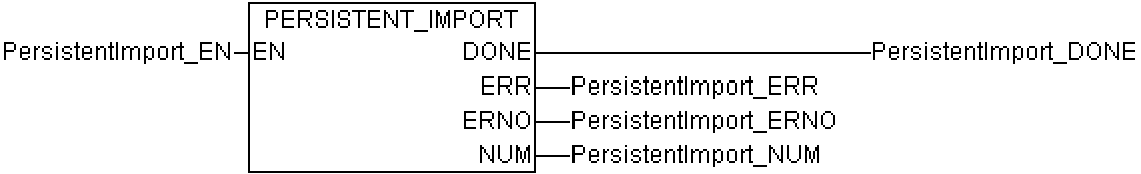
```
CAL  PERSISTENT_EXPORT(
     EN := PersistentExport_EN)

LD   PERSISTENT_EXPORT.DONE
ST   PersistentExport_DONE

LD   PERSISTENT_EXPORT.ERR
ST   PersistentExport_ERR

LD   PERSISTENT_EXPORT.ERNO
ST   PersistentExport_ERNO

LD   PERSISTENT_EXPORT.NUM
ST   PersistentExport_NUM
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
PERSISTENT_EXPORT (EN := PersistentExport_EN);

PersistentExport_DONE := PERSISTENT_EXPORT.DONE;
PersistentExport_ERR  := PERSISTENT_EXPORT.ERR;
PersistentExport_ERNO := PERSISTENT_EXPORT.ERNO;
PersistentExport_NUM  := PERSISTENT_EXPORT.NUM;
```

PERSISTENT_IMPORT Write persistent data from SD Card to RAM-DISC



With the function block PERSISTENT_IMPORT, all written data located in the PERSISTENT area or the %R area can be written from the SD Card to the RAM-DISC.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	OUTPUT	ARRAY [0..7] OF WORD	Error number of the corresponding segments, stored in an ARRAY [0..7]
NUM	Output	WORD	Number of written segments

Description

With a rising edge (False -> True) at input EN of the function block PERSISTENT_IMPORT, all data in the areas PERSISTENT or %R are written from the SD Card to the RAM-DISC.

The use of the function block requires that a valid PERSISTENT area or %R area is set in the PLC configuration of the CPU.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The ERNO output consists of an ARRAY [0..7], where the appeared error numbers of the segments are stored. The error messages encoding at output ERNO is explained at the beginning of the block description.

NUM WORD (number of areas)

The number of written segments is available at output NUM.

Function call in IL

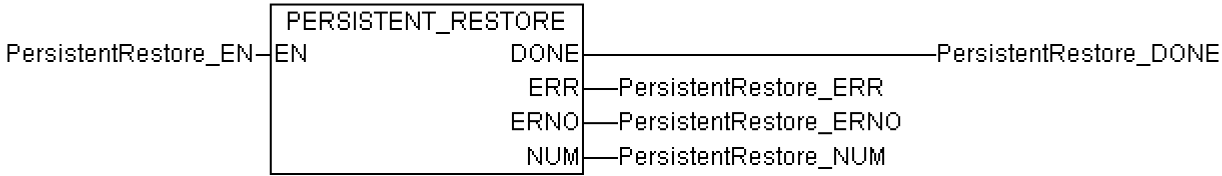
```
CAL  PERSISTENT_IMPORT(
     EN := PersistentImport_EN)

LD   PERSISTENT_IMPORT.DONE
ST   PersistentImport_DONE

LD   PERSISTENT_IMPORT.ERR
ST   PersistentImport_ERR

LD   PERSISTENT_IMPORT.ERNO
ST   PersistentImport_ERNO

LD   PERSISTENT_IMPORT.NUM
ST   PersistentImport_NUM
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
PERSISTENT_IMPORT (EN := PersistentImport_EN);

PersistentImport_DONE := PERSISTENT_IMPORT.DONE;
PersistentImport_ERR  := PERSISTENT_IMPORT.ERR;
PersistentImport_ERNO := PERSISTENT_IMPORT.ERNO;
PersistentImport_NUM  := PERSISTENT_IMPORT.NUM;
```

PERSISTENT_RESTORE Write persistent data from RAM-DISC to SRAM



With the function block PERSISTENT_RESTORE, all written data located in the PERSISTENT area or the %R area can be written from the RAM-DISC to the SRAM.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	OUTPUT	ARRAY [0..7] OF WORD	Error number of the corresponding segments, stored in an ARRAY [0..7]
NUM	Output	WORD	Number of written segments

Description

With a rising edge (False -> True) at input EN of the function block PERSISTENT_RESTORE, all data in the areas PERSISTENT or %R are written from the RAM-DISC to the SRAM.

The use of the function block requires that a valid PERSISTENT area or %R area is set in the PLC configuration of the CPU.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The ERNO output consists of an ARRAY [0..7], where the appeared error numbers of the segments are stored. The error messages encoding at output ERNO is explained at the beginning of the block description.

NUM WORD (number of areas)

The number of written segments is available at output NUM.

Function call in IL

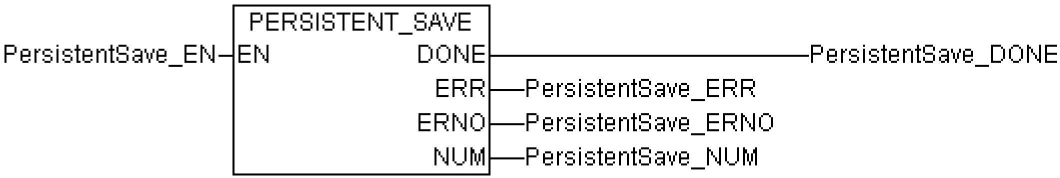
```
CAL   PERSISTENT_RESTORE(
      EN := PersistentRestore_EN)

LD    PERSISTENT_RESTORE.DONE
ST    PersistentRestore_DONE

LD    PERSISTENT_RESTORE.ERR
ST    PersistentRestore_ERR

LD    PERSISTENT_RESTORE.ERNO
ST    PersistentRestore_ERNO

LD    PERSISTENT_RESTORE.NUM
ST    PersistentRestore_NUM
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
PERSISTENT_RESTORE (EN := PersistentRestore_EN);

PersistentRestore_DONE := PERSISTENT_RESTORE.DONE;
PersistentRestore_ERR  := PERSISTENT_RESTORE.ERR;
PersistentRestore_ERNO := PERSISTENT_RESTORE.ERNO;
PersistentRestore_NUM  := PERSISTENT_RESTORE.NUM;
```

PERSISTENT_SAVE Write persistent data from SRAM to RAM-DISC



With the function block PERSISTENT_SAVE, all written data located in the PERSISTENT area or the %R area can be written from the SRAM to the RAM-DISC.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	OUTPUT	ARRAY [0..7] OF WORD	Error number of the corresponding segments, stored in an ARRAY [0..7]
NUM	Output	WORD	Number of written segments

Description

With a rising edge (False -> True) at input EN of the function block PERSISTENT_SAVE, all data in the areas PERSISTENT or %R are written from the SRAM to the RAM-DISC.

The use of the function block requires that a valid PERSISTENT area or %R area is set in the PLC configuration of the CPU.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The ERNO output consists of an ARRAY [0..7], where the appeared error numbers of the segments are stored. The error messages encoding at output ERNO is explained at the beginning of the block description.

NUM WORD (number of areas)

The number of written segments is available at output NUM.

Function call in IL

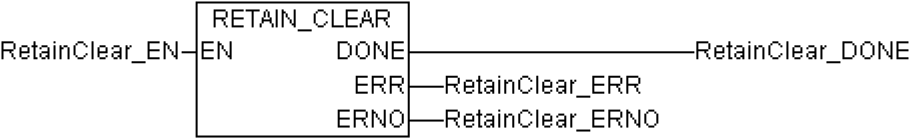
```
CAL  PERSISTENT_SAVE(
     EN := PersistentSave_EN)

LD   PERSISTENT_SAVE.DONE
ST   PersistentSave_DONE

LD   PERSISTENT_SAVE.ERR
ST   PersistentSave_ERR

LD   PERSISTENT_SAVE.ERNO
ST   PersistentSave_ERNO

LD   PERSISTENT_SAVE.NUM
ST   PersistentSave_NUM
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
PERSISTENT_SAVE (EN := PersistentSave_EN);

PersistentSave_DONE := PERSISTENT_SAVE.DONE;
PersistentSave_ERR  := PERSISTENT_SAVE.ERR;
PersistentSave_ERNO := PERSISTENT_SAVE.ERNO;
PersistentSave_NUM  := PERSISTENT_SAVE.NUM;
```

RETAIN_CLEAR Delete retain data from SRAM



With the function block RETAIN_CLEAR, all written retain data can be deleted from the SRAM.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

With a rising edge (False -> True) at input EN of the function block RETAIN_CLEAR, all data in the retain area are deleted.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

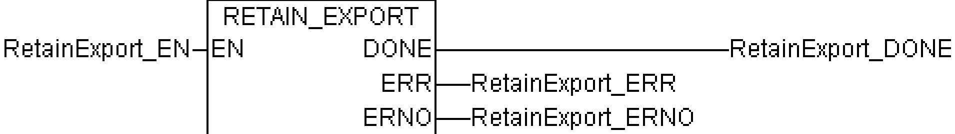
```
CAL   RETAIN_CLEAR(
      EN := RetainClear_EN)

LD    RETAIN_CLEAR.DONE
ST    RetainClear_DONE

LD    RETAIN_CLEAR.ERR
ST    RetainClear_ERR

LD    RETAIN_CLEAR.ERNO
ST    RetainClear_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
RETAIN_CLEAR (EN := RetainClear_EN);

RetainClear_DONE := RETAIN_CLEAR.DONE;
RetainClear_ERR  := RETAIN_CLEAR.ERR;
RetainClear_ERNO := RETAIN_CLEAR.ERNO;
```

RETAIN_EXPORT Write retain data from RAM-DISC to SD Card



With the function block RETAIN_EXPORT, all retain data can be written from the RAM-DISC to the SD Card.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

With a rising edge (False -> True) at input EN of the function block RETAIN_EXPORT, all data in the retain area are written from the RAM-DISC to the SD Card.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

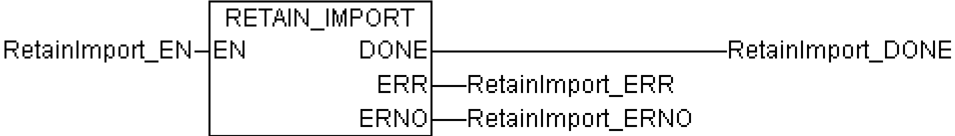
```
CAL   RETAIN_EXPORT(
      EN := RetainExport_EN)

LD    RETAIN_EXPORT.DONE
ST    RetainExport_DONE

LD    RETAIN_EXPORT.ERR
ST    RetainExport_ERR

LD    RETAIN_EXPORT.ERNO
ST    RetainExport_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
RETAIN_EXPORT (EN := RetainExport_EN);

RetainExport_DONE := RETAIN_EXPORT.DONE;
RetainExport_ERR  := RETAIN_EXPORT.ERR;
RetainExport_ERNO := RETAIN_EXPORT.ERNO;
```

RETAIN_IMPORT Write retain data from SD Card to RAM-DISC



With the function block RETAIN_IMPORT, all retain data can be written from the SD Card to the RAM-DISC.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

With a rising edge (False -> True) at input EN of the function block RETAIN_IMPORT, all data in the retain area are written from the SD Card to the RAM-DISC.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

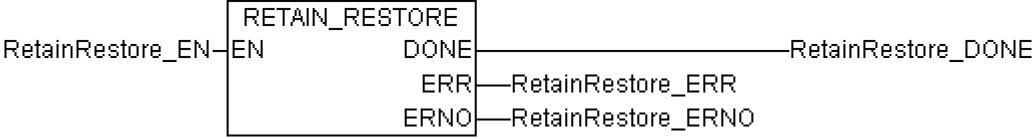
```
CAL   RETAIN_IMPORT(
      EN := RetainImport_EN)

LD    RETAIN_IMPORT.DONE
ST    RetainImport_DONE

LD    RETAIN_IMPORT.ERR
ST    RetainImport_ERR

LD    RETAIN_IMPORT.ERNO
ST    RetainImport_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
RETAIN_IMPORT (EN := RetainImport_EN);

RetainImport_DONE := RETAIN_IMPORT.DONE;
RetainImport_ERR  := RETAIN_IMPORT.ERR;
RetainImport_ERNO := RETAIN_IMPORT.ERNO;
```

RETAIN_RESTORE Write retain data from RAM-DISC to SRAM



With the function block RETAIN_RESTORE, all retain data can be written from the RAM-DISC to the SRAM.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

With a rising edge (False -> True) at input EN of the function block RETAIN_RESTORE, all data in the retain area are written from the RAM-DISC to the SRAM.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

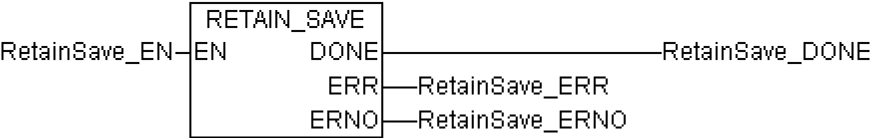
```
CAL   RETAIN_RESTORE(
      EN := RetainRestore_EN)

LD    RETAIN_RESTORE.DONE
ST    RetainRestore_DONE

LD    RETAIN_RESTORE.ERR
ST    RetainRestore_ERR

LD    RETAIN_RESTORE.ERNO
ST    RetainRestore_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
RETAIN_RESTORE (EN := RetainRestore_EN);

RetainRestore_DONE := RETAIN_RESTORE.DONE;
RetainRestore_ERR  := RETAIN_RESTORE.ERR;
RetainRestore_ERNO := RETAIN_RESTORE.ERNO;
```

RETAIN_SAVE Write retain data from SRAM to RAM-DISC



With the function block RETAIN_RESTORE, all retain data can be written from the SRAM to the RAM-DISC.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	SysInt_AC500_V10.LIB	

Block type

Function block without historical values (program)

Parameters

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

With a rising edge (False -> True) at input EN of the function block RETAIN_SAVE, all data in the retain area are written from the SRAM to the RAM-DISC.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

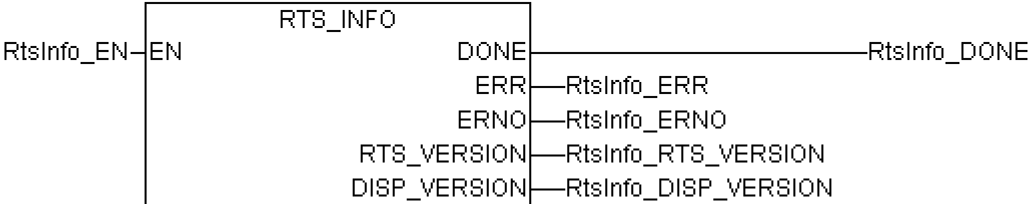
```
CAL   RETAIN_SAVE(
      EN := RetainSave_EN)

LD    RETAIN_SAVE.DONE
ST    RetainSave_DONE

LD    RETAIN_SAVE.ERR
ST    RetainSave_ERR

LD    RETAIN_SAVE.ERNO
ST    RetainSave_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
RETAIN_SAVE (EN := RetainSave_EN);

RetainSave_DONE := RETAIN_SAVE.DONE;
RetainSave_ERR  := RETAIN_SAVE.ERR;
RetainSave_ERNO := RETAIN_SAVE.ERNO;
```

RTS_INFO Reading the version of the CPU runtime system



The block RTS_INFO reads the version of the CPU runtime system.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block without historical values

Parameter

EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
RTS_VERSION	Output	WORD	Version of the CPU runtime system
DISP_VERSION	Output	WORD	Software version of the display

Description

Using the block RTS_INFO the version of the CPU runtime system can be read.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

RTS_VERSION WORD (RTS version)

RTS_VERSION outputs the version of the CPU runtime system. The upper BYTE of the entry represents the main version, the lower BYTE represents the subversion of the runtime system.

Example: RTS_VERSION = 16#0110 -> V01.16

DISP_VERSION WORD (display version)

DISP_VERSION outputs the software version of the display. The upper BYTE of the entry represents the main version, the lower BYTE represents the subversion of the display software.

Example: DISP_VERSION = 16#0110 -> V01.16

Function call in IL

```
CAL  RTS_INFO(
     EN := RtsInfo_EN)

LD   RTS_INFO.DONE
ST   RtsInfo_DONE

LD   RTS_INFO.ERR
ST   RtsInfo_ERR

LD   RTS_INFO.ERNO
ST   RtsInfo_ERNO

LD   RTS_INFO.RTS_VERSION
ST   RtsInfo_RTS_VERSION

LD   RTS_INFO.DISP_VERSION
ST   RtsInfo_DISP_VERSION
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
RTS_INFO (EN := RtsInfo_EN);

RtsInfo_DONE := RTS_INFO.DONE;
RtsInfo_ERR := RTS_INFO.ERR;
RtsInfo_ERNO := RTS_INFO.ERNO;
RtsInfo_RTS_VERSION := RTS_INFO.RTS_VERSION;
RtsInfo_DISP_VERSION := RTS_INFO.DISP_VERSION;
```

Structure of the file USRDATXX.DAT on the SD Card

Depending on the AC500 CPU type, the data are stored to the SD card in the following directory:

AC500 CPU	Directory	File
PM571	..\UserData\PM571\UserDat	USRDATxx.dat
PM581	..\UserData\PM581\UserDat	USRDATxx.dat
PM591	..\UserData\PM591\UserDat	USRDATxx.dat

An SD Card must be inserted in the AC500!

A maximum of 100 files (USRDAT0.DAT...USRDAT99.DAT) can be stored in one directory. Each data file USRDATxx.dat can be divided into individual sectors, if necessary. The "sector label" enclosed in square brackets (such as [Sector_01]<CR><LF>) indicates the start of the sector. Within a sector, data are saved as data sets in ASCII format. The individual elements of a data set are automatically separated by semicolon. Each data set is terminated with <CR><LF> ($0d_{hex}$, $0a_{hex}$).

This allows to directly import/export the data files from/to EXCEL. The data files can be viewed and edited using a standard ASCII editor (such as Notepad).

When saving / loading the data files, the following rules have to be observed:

- Writing on a non-existent file creates that file prior to the first write access.
- Data sets within a sector must always have the same number of values.
- Data sets in different sectors can have a different number of values.
- The values of a data set must have the same data format (BYTE, WORD, INT,..).
- A sector can have data sets with different data format. (Warning: The user must know the structure of the data when reading them.)
- The data sets are always appended to the end of the file when writing them.
- Searching for a "sector label" within a file is possible when reading it.
- Data sets can be read starting from a particular "sector label".
- A particular data set of a sector cannot be read or written.
- If you want to read each data set individually, a "sector label" must be inserted before each data set.
- Reading and writing the data with help of the user program is done with the blocks SD_READ and SD_WRITE.
- The values of a data set must be available in variables successively arranged in the PLC (e.g. ARRAY, STRING, %M area).
- A data file can be deleted with help of the PLC program.
- ndividual data sets and/or sectors cannot be deleted with the user program. This has to be done on the PC using an ASCII editor such as Notepad.

Data file examples:

Example 1:

Data file USRDAT5.dat without sectors:
-> 5 data sets, each with 10 DINT values:

600462;430;506;469;409;465;466;474;476;-1327203
600477;446;521;484;425;480;482;490;491;-1327187
600493;461;537;499;440;496;497;505;507;-1327172
600508;477;552;515;456;511;513;521;522;-1327156
600524;492;568;530;471;527;528;536;538;-1327141

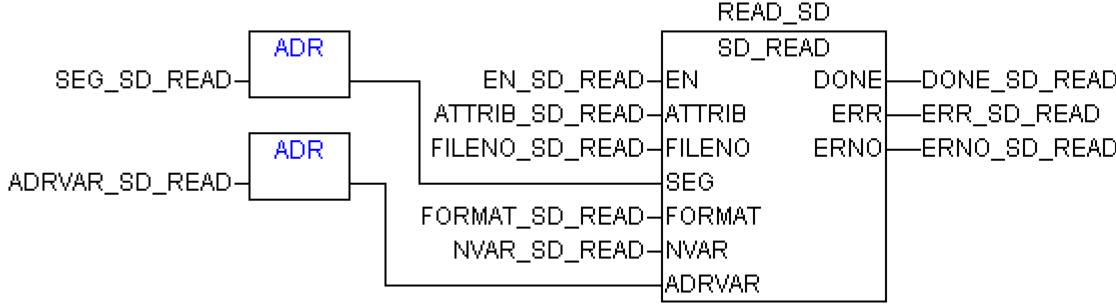
Example 2:

Data file USRDAT7.dat with sectors:
-> 3 sectors, each with 3 data sets and 10 DINT values per data set:

[Sector_01]
610439;10408;10483;10446;10387;10442;10444;10452;10453;-1317225
610455;10423;10499;10462;10402;10458;10460;10467;10469;-1317209
610476;10445;10520;10483;10424;10479;10481;10489;10490;-1317188
[Sector_02]
610570;10539;10614;10577;10518;10573;10575;10583;10584;-1317094
610585;10554;10630;10592;10533;10589;10591;10598;10600;-1317078
610602;10571;10646;10609;10550;10605;10607;10615;10616;-1317062
[Sector_03]
610701;10670;10746;10708;10649;10704;10706;10714;10715;-1316963
610717;10686;10761;10724;10665;10720;10722;10730;10731;-1316947
610739;10708;10783;10746;10686;10742;10744;10751;10753;-1316926

The sector name can consist of a maximum of 32 characters (including []).

SD_READ Reading a data set from the SD Card



The function block SD_READ reads a data set from a file on the SD Card and stores the read data set beginning at the start flag defined by ADRVAR.

In this context it has to be observed that the blocks are mutually interlocked, i.e. it must be ensured that only one block is active at the same time.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block with historical values

Parameters

Instance		SD_READ	Instance name
EN	Input	BOOL	Reading a data set is triggered by a FALSE/TRUE edge
ATTRIB	Input	BYTE	Read attribute of the block: 1 - Open file, search sector, read data set (Open, Seek, Read)
FILENO	Input	BYTE	Number of the storage file (XX = 0 to 99) USRDATXX.DAT
SEG	Input	DWORD	Address of the segment name
FORMAT	Input	BYTE	Format of the data set elements Data format: 00hex - 0 - BYTE 01 hex - 1 - CHAR 10 hex - 16 - WORD 11 hex - 17 - INT 20 hex - 32 - DWORD 21 hex - 33 - DINT
NVAR	Input	WORD	Number of elements per data set
ADRVAR	Input	DWORD	Start target address of the data set
DONE	Output	BOOL	Read operation completed
ERR	Output	BOOL	Error occurred
ERNO	Output	WORD	Error number

Description

The function block reads a data set in the file on the SD card:
...\UserData\PM5x1\UserDat\USRDATxx.DAT (see also "Structure of the file USRDATXX.DAT on the SD Card")

Important note:

Access to the SD Card is only possible by using the function blocks SD_WRITE and SD_READ.

The inputs ATTRIB, FILENO, FORMAT, ADRVAR and NVAR determine how many values should be read from which file and in which format on the SD card as well as to which target address they should be stored. Always a complete data set must be read.

Reading a data set from the SD Card can take several PLC cycles.

With a FALSE/TRUE edge at input EN, the data set reading is triggered once. Input EN is not evaluated again until the ready message DONE = TRUE is available, i.e. the state of EN is ignored during reading.

After the read operation is completed, the block outputs DONE, ERR and ERNO are updated. Reading was successful, if DONE = TRUE and ERR = FALSE. If DONE = TRUE and ERR = TRUE, an error occurred. The error type is signalized at output ERNO.

After reading a data set from the SD Card, the block outputs are valid for one cycle. In the next cycle, the outputs DONE, ERR and ERNO are reset to zero. A new FALSE/TRUE edge at input EN starts a new read operation.

Reading example 1:

To read user data from a data file without sectors from the SD card and write them to the PLC, proceed as follows:

1. Insert the SD card.

2. Read a data set by calling the block SD_READ with the following settings:

   EN := TRUE (* FALSE/TRUE edge triggers reading *)
   ATTRIB := 2 (* open / read *)
   FILENO := 0..99 (* number of file to be read *)
   SEG := address of the variable sector name
   FORMAT := data format
   NVAR := number of data in the data set
   ADRVAR := address of the first variable to which data are to be stored.

3. Further data sets can be read with the following settings after the completion message (DONE=TRUE) is displayed. This process is started with a FALSE/TRUE edge at input EN:

   EN := TRUE (* FALSE/TRUE edge triggers reading*)
   ATTRIB := 3 (* continue read *)
   FILENO := 0..99 (* number of file to be read *)
   SEG := address of the variable sector name
   FORMAT := data format
   NVAR := number of data in the data set
   ADRVAR := address of the first variable to which data are to be stored

   If an unexpected sector name or the end of file (EOF) is detected during reading, an appropriate error message is generated.

4. To read a further data set and to close the file afterwards, call the block SD_READ with the following settings after the completion message (output DONE=TRUE) is displayed and start the process with a FALSE/TRUE edge at input EN:
   EN := TRUE (* FALSE/TRUE edge triggers reading*)
   ATTRIB := 4 (* read / close *)
   FILENO := 0...99 (* number of file to be read *)
   SEG := address of the variable sector name
   FORMAT := data format

NVAR := number of data in the data set
ADRVAR := address of the first variable to which data are to be stored

If an unexpected sector name or the end of file (EOF) is detected during reading, an appropriate error message is generated.

5. To close the file without reading it, call the block SD_READ with the following settings after the completion message (DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:
   EN := TRUE (* FALSE/TRUE edge closes the file *)
   ATTRIB := 5 (* close *)
   FILENO := 0..99 (* number of file to be closed *)

Reading example 2:

To read user data from a data file with sectors from the SD card and write them to the PLC, proceed as follows:

1. Insert the SD card.

2. Seek a sector label and read a data set by calling the block SD_READ with the following settings:
   EN := TRUE (* FALSE/TRUE edge triggers reading *)
   ATTRIB := 1 (* open / seek / read *)
   FILENO := 0..99 (* number of the file to be read *)
   SEG := address of the variable sector name
   FORMAT := data format
   NVAR := number of data in the data set
   ADRVAR := address of the first variable to which data should be written

   The read operation is finished successfully if output DONE = TRUE and output ERR = FALSE. A seek error is indicated with ERR = TRUE and ERNO <> 0.

3. Further data sets can be read with the following settings after the completion message (DONE=TRUE) is displayed. This process is started with a FALSE/TRUE edge at input EN:
   EN := TRUE (* FALSE/TRUE edge triggers reading*)
   ATTRIB := 3 (* continue read *)
   FILENO := 0..99 (* number of file to be read *)
   SEG := address of the variable sector name
   FORMAT := data format
   NVAR := number of data in the data set
   ADRVAR := address of the first variable to which data are to be stored

   If an unexpected sector name or the end of file (EOF) is detected during reading, an appropriate error message is generated.

4. If you want to read further sectors / data sets, repeat steps 2 and 3.

5. To read a further data set and to close the file afterwards, call the block SD_READ with the following settings after the completion message (output DONE=TRUE) is displayed and start the process with a FALSE/TRUE edge at input EN:
   EN := TRUE (* FALSE/TRUE edge triggers reading*)
   ATTRIB := 4 (* read / close *)
   FILENO := 0..99 (* number of file to be read *)
   SEG := address of the variable sector name
   FORMAT := data format
   NVAR := number of data in the data set
   ADRVAR := address of the first variable to which data are to be written

   If an unexpected sector name or the end of file (EOF) is detected during reading, an appropriate error message is generated.

6. To close the file without reading it, call the block SD_READ with the following settings after the completion message (DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:
   EN := TRUE (* FALSE/TRUE edge closes the file *)
   ATTRIB := 5 (* close *)
   FILENO := 0..99 (* number of file to be closed *)

EN BOOL (enable)

Processing of the block is controlled via input EN.

EN = FALSE:

The outputs DONE, ERR and ERNO are set to "0" or FALSE, respectively. This is not valid during reading.

EN = FALSE/TRUE edge:

Reading of the data set is started once.

ATTRIB BYTE (attribute)

At input ATTRIB, the block operation (action) is specified.

Possible values:

1 – Open file, search sector, read data set (Open, Seek, Read), additionally needed inputs:

FILENO, SEG, FORMAT, NVAR, ADRVAR

2 – Open file, read data set (Open, Read), additionally needed inputs:

FILENO, FORMAT, NVAR, ADRVAR

3 – Read next data set (Read), additionally needed inputs:

FILENO, FORMAT, NVAR, ADRVAR

4 – Read data set, close file (Read, Close), additionally needed inputs:

FILENO, FORMAT, NVAR, ADRVAR

5 – Close file (Close), additionally needed inputs:

FILENO

SEG DWORD (segment)

At input SEG, the start address of the segment label to be searched is specified. A segment label must be enclosed in brackets "[...]".

Examples:

[Values_Tab1]

[Temperature_12]

The length is limited to 32 characters.

FILENO BYTE (file number)

At input FILENO, the number of the file is specified from which data are to be read. Valid values: 0...99

FORMAT BYTE (format)

Input Format is used to define the format of the data elements. All elements of one data set must have the same format.

Valid data formats:
00 hex - 0 - BYTE
01 hex - 1 - CHAR
10 hex - 16 - WORD
11 hex - 17 - INT
20 hex - 32 - DWORD
21 hex - 33 - DINT

NVAR WORD (number of variable)

At input NVAR, the number of elements of the data set to be read is specified.

ADRVAR DWORD (address of variable)

Input ADRVAR is used to specify the target start address of the data set. The values of a data set are stored in variables successively arranged in the PLC (e.g. ARRAY, STRING, %M area).

DONE BOOL (done)

Output DONE indicates that the read operation of the data set is completed. This output has always to be considered together with output ERR.

The following applies:

DONE = TRUE and ERR = FALSE: The read operation is completed. The data set has been read successfully from the file on the SD Card.

DONE = TRUE and ERR = TRUE: An error occurred while reading the data set. Output ERNO signalizes the error number.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the read operation. This output always has to be considered together with output DONE. If an error has occurred, the following applies: DONE = TRUE and ERR = TRUE. Output ERNO signalizes the error number.

ERNO WORD (error number)

The output ERNO indicates an error number. This output has always to be considered together with the outputs DONE and ERR.

Function call in IL

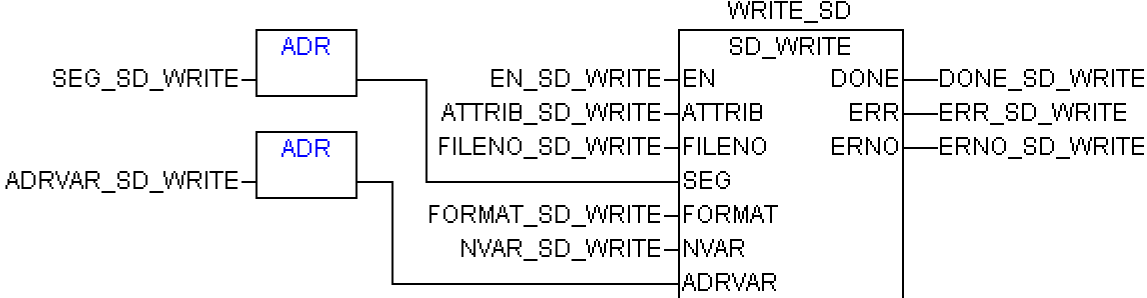
```
CAL  READ_SD(EN := EN_SD_READ,
     ATTRIB := ATTRIB_SD_READ,
     FILENO := FILENO_SD_READ,
     SEG := SEG_SD_READ,
     FORMAT := FORMAT_SD_READ,
     NVAR := NVAR_SD_READ,
     ADRVAR := ADRVAR_SD_READ)

LD   READ_SD.DONE
ST   DONE_SD_READ

LD   READ_SD.ERR
ST   ERR_SD_READ

LD   READ_SD.ERNO
ST   ERNO_SD_READ
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
READ_SD(EN := EN_SD_READ,
     ATTRIB := ATTRIB_SD_READ,
     FILENO := FILENO_SD_READ,
     SEG := SEG_SD_READ,
     FORMAT := FORMAT_SD_READ,
     NVAR := NVAR_SD_READ,
     ADRVAR := ADRVAR_SD_READ);
DONE_SD_READ := READ_SD.DONE;
ERR_SD_READ := READ_SD.ERR;
ERNO_SD_READ := READ_SD.ERNO;
```

SD_WRITE Writing a data set to the SD Card



The AC500 control system contains a memory card of the type "SD Memory Card" (in short SD card) as external storage medium which is accessed by the PLC like a floppy disk drive. The SD card is used to transfer data between a commercially available PC with SD card interface and the AC500 control system.

Read and write accesses take quite long time, since they are handled in the background by the internal file system of the operating system. When performing a write access, the current file is always copied to a backup file.

In this context it has to be observed that the blocks are mutually interlocked, i.e. it must be ensured that only one block is active at the same time.

The function block SD_WRITE writes a data set to a file USRDATxx.DAT on the SD Card.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block with historical values

Parameter

Instance		SD_WRITE	Instance name
EN	Input	BOOL	Writing a data set is triggered by a FALSE/TRUE edge
ATTRIB	Input	BYTE	Write attribute of the block: 1 - Delete file (Delete) 2 - Write data set (Open(create), Write(append), Close) 3 - Write segment label (Open(create), Write(append), Close)
FILENO	Input	BYTE	Number of the storage file (XX = 0 to 99) USRDATXX.DAT
SEG	Input	DWORD	Address of the segment name
FORMAT	Input	BYTE	Format of the data set elements Data format: 00hex - 0 - BYTE 01 hex - 1 - CHAR 10 hex - 16 - WORD 11 hex - 17 - INT 20 hex - 32 - DWORD 21 hex - 33 - DINT

NVAR	Input	WORD	Number of elements per data set
ADRVAR	Input	DWORD	Start source address of the data set
DONE	Output	BOOL	Write operation completed
ERR	Output	BOOL	Error has occurred
ERNO	Output	WORD	Error number

Description

The function block writes a data set to a file on the SD card:
...\UserData\PM5x1\UserDat\USRDATxx.DAT (see also "Structure of the file USRDATXX.DAT on the SD Card")

Important note:

Access to the SD Card is only possible by using the function blocks SD_WRITE and SD_READ.

The inputs ATTRIB, FILENO, FORMAT, ADRVAR and NVAR determine how many values should be written to which file and in which format on the SD card as well as from which source address they should be read. To create a readable and EXCEL-compatible file, the individual values are stored in ASCII format, automatically separated by a semicolon. The last value is automatically terminated by a <CR><LF>.

When a write operation of a data set is started (FALSE/TRUE edge at input EN), the data of the data set must not be changed until the end of the write procedure (DONE = TRUE). Storing a data set on the SD Card can take several PLC cycles.

Input EN is not evaluated again until the ready message DONE = TRUE is available, i.e. the state of EN is ignored during writing.

After the write operation is completed, the block outputs DONE, ERR and ERNO are updated. The storage was successful, if DONE = TRUE and ERR = FALSE. If DONE = TRUE and ERR = TRUE, an error occurred. The error type is signalized at output ERNO.

After storing a data set on the SD Card, the block outputs are valid for one cycle. In the next cycle, the outputs DONE, ERR and ERNO are reset to zero. A new FALSE/TRUE edge at input EN starts a new write operation.

Note:

In case of a power failure during the write access, the file USRDATxx.DAT will be corrupted. In order to backup at least the already stored data sets, the file USRDATxx.BAK has to be copied from the SC Card prior to restarting the program. The file can be renamed to USRDATxx.DAT on the PC and can then be used for further storage.

Writing example 1:

To store user data to the SD card in a data file without sectors, proceed as follows:

1. Insert the SD card.

2. Write a data set by calling the block SD_WRITE with the following settings:
   EN := TRUE (* FALSE/TRUE edge triggers write operation *)
   ATTRIB := 2 (* write append *)
   FILENO := 0..99 (* number of the file to be written *)
   SEG := address of the variable sector name (* any *)
   FORMAT := data format
   NVAR := number of data in the data set
   ADRVAR := address of the first variable to be written

   If no appropriate file can be found, it will be created.
   The write process is successfully completed if output DONE = TRUE and output ERR = FALSE. A write error is indicated with ERR = TRUE and ERNO <> 0.

3. Further data sets can be written with the same block settings after the completion message (output DONE=TRUE) is displayed. This process is started with a FALSE/TRUE edge at input EN.

Writing example 2:

To store user data to the SD card in a data file with sectors, proceed as follows:

1. Insert the SD card.

2. Write the sector label by calling the block SD_WRITE with the following settings:
   EN := TRUE
   ATTRIB := 3 (* write sector *)
   FILENO := 0..99 (* number of the file to be written *)
   SEG := address of the variable sector name

   If no appropriate file can be found, it will be created.
   The sector is successfully written when the output DONE:=TRUE and the output ERR:=FALSE. A write error is indicated with ERR = TRUE and ERNO <> 0.

3. Write a data set by calling the block SD_WRITE with the following settings:
   EN := TRUE (* FALSE/TRUE edge triggers write operation *)
   ATTRIB := 2 (* write append *)
   FILENO := 0..99 (* number of the file to be written *)
   SEG := address of the variable sector name
   FORMAT := data format
   NVAR := number of data in the data set
   ADRVAR := address of the first variable to be written

   The write process is successfully completed if output DONE = TRUE and output ERR = FALSE. A write error is indicated with ERR = TRUE and ERNO <> 0.

4. Further data sets can be written with the same block settings after the completion message (output DONE=TRUE) is displayed. This process is started with a FALSE/TRUE edge at input EN.

5. If you want to write further sectors and data sets, repeat steps 2...4.

   Note:

   The file USRDATxx.DAT is saved as USRDATxx.BAK for each write process and a "Open file / Write file / Close file" procedure is performed.

Deleting a file:

To delete a data file from the SD card, proceed as follows:

1. Insert the SD card.

2. Call the block SD_WRITE with the following settings:
   EN := TRUE
   ATTRIB := 1 (* delete *)
   FILENO := 0..99 (* number of the file to be deleted *)
   SEG, FORMAT, NVAR, ADRVAR – any

EN BOOL (enable)

Processing of the block is controlled via input EN.

EN = FALSE:

The outputs DONE, ERR and ERNO are set to "0" or FALSE, respectively. This is not valid during writing.

EN = FALSE/TRUE edge:

Writing of the data set / segment label or deletion of the file respectively is started once.

ATTRIB BYTE (attribute)

At input ATTRIB, the block operation (action) is specified.

Possible values:

1 – Delete file (Delete), additionally needed inputs:

FILENO

2 – Write data set (Open(create), Write(append), Close), additionally needed inputs:

FILENO, FORMAT, NVAR, ADRVAR

3 – Write segment label (Open(create), Write(append), Close), additionally needed inputs:

FILENO, SEG

SEG DWORD (segment)

At input SEG, the start address of the segment label is specified. A segment label must be enclosed in brackets "[...]".

Examples:

[Values_Tab1]

[Temperature_12]

The length is limited to 32 characters.

FILENO BYTE (file number)

At input FILENO, the number of the file is specified to which data are to be written or which should be created or deleted respectively.

Valid values: 0 ... 99

FORMAT BYTE (format)

Input Format is used to define the format of the data elements. All elements of one data set must have the same format.

Valid data formats:
00 hex - 0 - BYTE
01 hex - 1 - CHAR
10 hex - 16 - WORD
11 hex - 17 - INT
20 hex - 32 - DWORD
21 hex - 33 - DINT

NVAR WORD (number of variable)

At input NVAR, the number of elements of a data set is specified.

ADRVAR DWORD (address of variable)

Input ADRVAR is used to specify the start address of the data set. The values of a data set must be available in variables successively stored in the PLC (e.g. ARRAY, STRING, %M area).

DONE BOOL (done)

Output DONE indicates that the write operation of the data set is completed. This output has always to be considered together with output ERR.

The following applies:

DONE = TRUE and ERR = FALSE: The write operation is completed. The data set has been stored successfully in the file on the SD Card.

DONE = TRUE and ERR = TRUE: An error occurred during the write operation. Output ERNO signalizes the error number.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the write operation. This output always has to be considered together with output DONE. If an error has occurred, the following applies: DONE = TRUE and ERR = TRUE. Output ERNO signalizes the error number.

ERNO WORD (error number)

The output ERNO indicates an error number. This output has always to be considered together with the outputs DONE and ERR.

Function call in IL

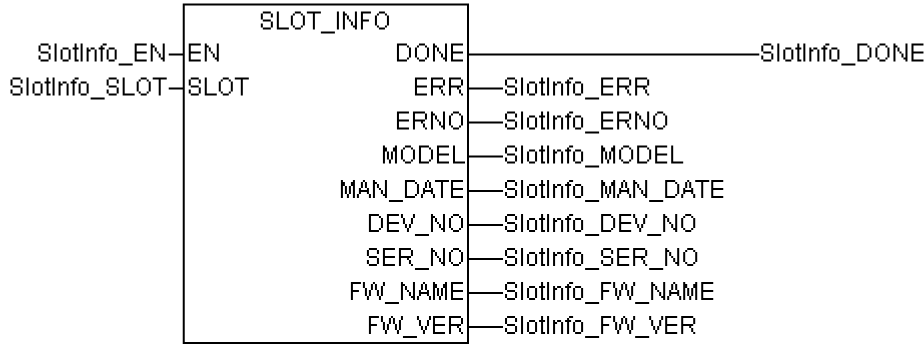
```
CAL  WRITE_SD(EN := EN_SD_WRITE,
     ATTRIB := ATTRIB_SD_WRITE,
     FILENO := FILENO_SD_WRITE,
     SEG := SEG_SD_WRITE,
     FORMAT := FORMAT_SD_WRITE,
     NVAR := NVAR_SD_WRITE,
     ADRVAR := ADRVAR_SD_WRITE)

LD   WRITE_SD.DONE
ST   DONE_SD_WRITE

LD   WRITE_SD.ERR
ST   ERR_SD_WRITE

LD   WRITE_SD.ERNO
ST   ERNO_SD_WRITE
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
WRITE_SD(EN := EN_SD_WRITE,
     ATTRIB := ATTRIB_SD_WRITE,
     FILENO := FILENO_SD_WRITE,
     SEG := SEG_SD_WRITE,
     FORMAT := FORMAT_SD_WRITE,
     NVAR := NVAR_SD_WRITE,
     ADRVAR := ADRVAR_SD_WRITE);
DONE_SD_WRITE := WRITE_SD.DONE;
ERR_SD_WRITE := WRITE_SD.ERR;
ERNO_SD_WRITE := WRITE_SD.ERNO;
```

SLOT_INFO Reading slot information



The block SLOT_INFO reads information from the slot.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function block without historical values

Parameter

EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (card number)
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
MODEL	Output	STRING(24)	Device type
MAN_DATE	Output	DATE	Manufacturing date
DEV_NO	Output	DWORD	Device number
SER_NO	Output	DWORD	Serial number
FW_NAME	Output	STRING(16)	Firmware name
FW_VER	Output	STRING(16)	Firmware version

Description

By means of the SLOT_INFO block, slot information can be read from the connected device (coupler).

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

SLOT BYTE (slot)

At the input SLOT, the slot (card number) is selected, from which the information is to be read.

The internal slot always has the number 0. The external slots are counted from right to left and begin with the number 1.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

MODEL STRING(24) (model)

The output MODEL provides the type of the device which is connected to the selected slot. The output is in plain text.

MAN_DATE DATE (manufacture date)

The output MAN_DATE provides the manufacturing date of the device which is connected to the selected slot.

DEV_NO DWORD (device number)

The output DEV_NO provides the manufacturer-specific number of the device which is connected to the selected slot.

SER_NO DWORD (serial number)

The output SER_NO provides the serial number of the device which is connected to the selected slot.

FW_NAME STRING(16) (firmware name)

The output FW_NAME provides the firmware name of the device which is connected to the selected slot. The output is in plain text.

FW_VER STRING(16) (firmware version)

The output FW_VER provides the firmware version of the device which is connected to the selected slot. The output is in plain text.

Function call in IL

```
CAL  SLOT_INFO(
     EN := SlotInfo_EN,
     SLOT := SlotInfo_SLOT)

LD   SLOT_INFO.DONE
ST   SlotInfo_DONE

LD   SLOT_INFO.ERR
ST   SlotInfo_ERR

LD   SLOT_INFO.ERNO
ST   SlotInfo_ERNO

LD   SLOT_INFO.MODEL
ST   SlotInfo_MODEL

LD   SLOT_INFO.MAN_DATE
ST   SlotInfo_MAN_DATE

LD   SLOT_INFO.DEV_NO
ST   SlotInfo_DEV_NO

LD   SLOT_INFO.SER_NO
ST   SlotInfo_SER_NO

LD   SLOT_INFO.FW_NAME
ST   SlotInfo_FW_NAME

LD   SLOT_INFO.FW_VER
ST   SlotInfo_FW_VER
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
SLOT_INFO (EN := SlotInfo_EN, SLOT := SlotInfo_SLOT);

SlotInfo_DONE := SLOT_INFO.DONE;
SlotInfo_ERR := SLOT_INFO.ERR;
SlotInfo_ERNO := SLOT_INFO.ERNO;
SlotInfo_MODEL := SLOT_INFO.MODEL;
SlotInfo_MAN_DATE := SLOT_INFO.MAN_DATE;
SlotInfo_DEV_NO := SLOT_INFO.DEV_NO;
SlotInfo_SER_NO := SLOT_INFO.SER_NO;
SlotInfo_FW_NAME := SLOT_INFO.FW_NAME;
SlotInfo_FW_VER := SLOT_INFO.FW_VER;
```

SYS_TIME Reading the system time

```
              SYS_TIME
SysTime_EN─EN                ───SysTime
```

The block SYS_TIME outputs the system tick in milliseconds as a double word.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	SysInt_AC500_V10.lib	

Block type

Function

Parameter

EN	Input	BOOL	Enabling of the block processing
	Output	DWORD	System tick in milliseconds

Description

The block SYS_TIME outputs the system tick as a double word. The system tick is provided with a resolution of a millisecond and also serves as a time basis for the PLC application program and all time-dependent blocks. After a PLC reset the system tick always starts with 0. An overflow is reached after 49 days. After this, the counter restarts at 0.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

(Output) DWORD

The block output delivers the system tick in ms.

Function call in IL

```
LD    SysTime_EN
SYS_TIME
ST    SysTime
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
SysTime := SYS_TIME(SysTime_EN);
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

COM_SET_PROT Setting/changing COM protocol actively 6

Components of the library 3

CPU_INFO Reading the CPU type 9

D

DIAG_ACK Acknowledging an error 11

DIAG_ACK_ALL Acknowledging all errors of an error class 14

DIAG_EVENT Generating an error event 16

DIAG_GET Reading an error 19

DIAG_INFO Displaying an overview of all errors not yet read 24

F

FLASH_DEL Deleting a data segment in Flash memory 26

FLASH_READ Reading a data set from Flash memory 29

FLASH_WRITE Writing a data set to Flash memory 33

FPU_EXCEPTION_INFO Reading information about the FPU exception 37

G

Glossary 90

I

IO_DIAG Reading the diagnosis data of the I/O bus 40

IO_INFO Reading the number of devices connected to the I/O bus 43

I/O_MODULE_DIAG Reading the module diagnosis data of the I/O bus 45

I/O_VERSION Reading the version of the I/O bus driver 48

O

Overview of blocks arranged according to their call names 5

P

PERSISTENT_CLEAR Delete persistent data from SRAM 50

PERSISTENT_EXPORT Write persistent data from RAM-DISC to SD Card 52

PERSISTENT_IMPORT Write persistent data from SD Card to RAM-DISC 54

PERSISTENT_RESTORE Write persistent data from RAM-DISC to SRAM 56

PERSISTENT_SAVE Write persistent data from SRAM to RAM-DISC 58

Preconditions for the use of the library 3

R

RETAIN_CLEAR Delete retain data from SRAM 60

RETAIN_EXPORT Write retain data from RAM-DISC to SD Card 62

RETAIN_IMPORT Write retain data from SD Card to RAM-DISC 64

RETAIN_RESTORE Write retain data from RAM-DISC to SRAM 66

RETAIN_SAVE Write retain data from SRAM to RAM-DISC 68

RTS_INFO Reading the version of the CPU runtime system 70

S

SD_READ Reading a data segment from the SD card 74

SD_WRITE Writing a data segment to the SD card 80

SLOT_INFO Reading the slot information 86

Special characteristics of the internal system library 3

Structure of the file USRDATXX.DAT stored on the SD card 72

SYS_TIME Reading the system time 89

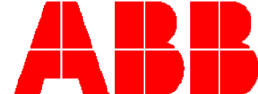
Software Description

AC500

Scalable PLC
for Individual Automation

Modbus
Function Block Library

Modbus



Contents

Modbus Library 2

Special characteristics of the Modbus Library 2

Components of the Modbus Library 2

Overview of blocks arranged according to their call names 2

COM_MOD_MAST Processing of Modbus master telegrams 3

Glossary 7

Index 9

Modbus Library

Special characteristics of the Modbus Library

Components of the Modbus Library

The Modbus library contains the following function blocks:

Group: Modbus	
COM_MOD_MAST	Processing of Modbus master telegrams

Overview of blocks arranged according to their call names

Used abbreviations:

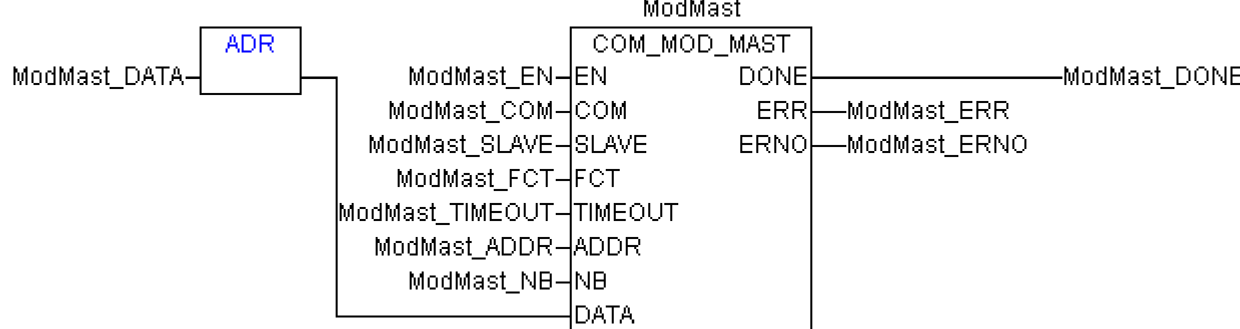
FBhv . .. Function block with historical values

FBnohv ... Function block without historical values

F … Function

VE name	Type	Function
COM_MOD_MAST	FBhv	Processing of Modbus master telegrams

COM_MOD_MAST Processing of Modbus master telegrams



The block COM_MOD_MAST realizes the MODBUS master function for the MODBUS interface (COM1, COM2) specified at input COM. For each interface, a separate COM_MOD_MAST block must be used. Prior to the use of COM_MOD_MAST for an interface, the particular interface has to be configured via the controller configuration of the Control Builder (PS501) (see System Technology / CPU controller configuration) as the MODBUS master interface. With each FALSE > TRUE edge at input EN, the function block COM_MOD_MAST reads the values at the inputs, generates a telegram according to the inputs and sends this telegram to the slave.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Modbus_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		COM_MOD_MAST	Instance name
EN	Input	BOOL	Enabling of the block processing
COM	Input	BYTE	Interface identifier (COM1, COM2)
SLAVE	Input	BYTE	Slave address
FCT	Input	BYTE	Function code
TIMEOUT	Input	WORD	Telegram timeout in ms
ADDR	Input	WORD	Operand/register address in the slave
NB	Input	WORD	Number of data
DATA	Input	DWORD	Address of the first operand area in the master, from which data are to be sent to the slave or data read by the slave should be stored.
DONE	Output	BOOL	Ready message
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block COM_MOD_MAST realizes the MODBUS master function for the MODBUS interface of the controller (COM1, COM2) specified at input COM.

For each interface, a separate COM_MOD_MAST block must be used.

Prior to the use of COM_MOD_MAST for an interface, the particular interface has to be configured via the controller configuration of the Control Builder (PS501) (see System Technology / CPU controller configuration) as the MODBUS master interface.

With each FALSE > TRUE edge at input EN, the function block COM_MOD_MAST reads the values at the inputs, generates a telegram according to the inputs and sends this telegram to the slave.

EN BOOL (enable)

If a FALSE > TRUE edge is applied to input EN, all further inputs are read in.

If the input values are valid, a request telegram is sent to a slave.

If at least one input is invalid, no telegram is generated and the error is displayed at the outputs ERR and ERNO instead.

While the request is processed, state changes at input EN are recognized but not evaluated.

COM BYTE (com)

At input COM, the MODBUS interface number is specified.

COM = 1: COM1
COM = 2: COM2

SLAVE BYTE (slave)

At input SLAVE, the address of the slave to which the telegram shall be sent is specified.

FCT BYTE (function code)

The function code of the request telegram is specified at input FCT.

01 or 02	read n bits
03 or 04	read n words
05	write one bit
06	write one word
07	read 8 bit
15	write n bits
16	write n words

TIMEOUT WORD (timeout)

The telegram timeout in milliseconds (ms) is specified at input TIMEOUT.

If no response is received within the time interval specified in TIMEOUT, the procedure is aborted and an error identifier is output.

Note: Keeping the timeout depends on the cycle time of the task in which the MODMAST block is processed. The real time may deviate from the specification in worst case by task cycle time - 1 ms.

ADDR WORD (address)

The operand/register address in the slave from which data should be read or written is specified at input ADDR.

The access to operands of AC500 devices in MODBUS slave mode is defined via the MODBUS cross-reference list. Only operands that are listed in the cross-reference list may be used (see Modbus address tables).

Only operands that are listed in the MODBUS address list may be used. When accessing other devices, ADDR is freely selectable. The valid ranges have to be gathered from the corresponding device description.

NB WORD (number)

At input NB, the number of data to be written or read is specified.

The unit of NB depends on the selected function. For bit accesses the number of bits, for word and double word accesses the number of words is specified at NB.

The following restrictions apply to the length:

FCT	Nb_{max}
01 or 02	2000 bits
03 or 04	125 words / 62 double words
05	1 bit
06	1 word
07	8 bits
15	1968 bits
16	123 words / 61 double words

DATA DWORD (data)

At input DATA, the address of the first operand in the master is specified, from which data are copied/written to the slave or to which the data read by the slave should be stored. For this purpose it is necessary that the operand type (e.g. bit) matches the selected function (e.g. FCT 1, read n bits).

DONE BOOL (done)

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

```
LD   ModMast_DATA
ADR
ST   ModMast.DATA

CAL  ModMast (
     EN      := ModMast_EN,
     COM     := ModMast_COM,
     SLAVE   := ModMast_SLAVE,
     FCT     := ModMast_FCT,
     TIMEOUT := ModMast_TIMEOUT,
     ADDR    := ModMast_ADDR,
     NB      := ModMast_NB,
     DATA    := ModMast_DATA)

LD   ModMast.DONE
ST   ModMast_DONE

LD   ModMast.ERR
ST   ModMast_ERR

LD   ModMast.ERNO
ST   ModMast_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
ModMast  (EN      := ModMast_EN,
          COM     := ModMast_COM,
          SLAVE   := ModMast_SLAVE,
          FCT     := ModMast_FCT,
          TIMEOUT := ModMast_TIMEOUT,
          ADDR    := ModMast_ADDR,
          NB      := ModMast_NB,
          DATA    := ADR(ModMast_DATA));

ModMast_DONE := ModMast.DONE;
ModMast_ERR := ModMast.ERR;
ModMast_ERNO:= ModMast.ERNO;
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

COM_MOD_MAST Processing of the Modbus master telegrams 3

Components of the Modbus library 2

G

Glossary 7

O

Overview of blocks arranged according to their call names 2

S

Special characteristics of the Modbus library 2

Software Description

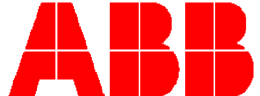
AC500

Scalable PLC
for Individual Automation

Series90 AC500
Function Block Library

Series90

AC500



Contents

Series90 AC500 Library 2

Components of the library 2

Overview of blocks arranged according to their call names 2

INTK Integrator (extended) 3

LZB Run number block 7

VGL3P Comparator with 3-point response 9

Glossary 11

Index 13

Series90 AC500 Library

Components of the Series90 AC500 library

The Series90 AC500 library contains the following function blocks:

INTK	Integrator (extended)
LZB	Run number block
VGL3P	Comparator with 3-point response

Overview of blocks arranged according to their call names

Character description:

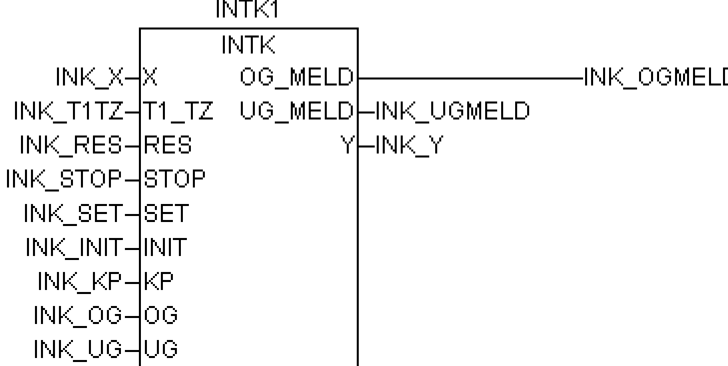
FBhV … Function block with historical values

FBnohV … Function block without historical values

F … Function

VE name	Type	Function
INTK	FBhV	Integrator (extended)
LZB	FBhV	Run number block
VGL3P	FBnohV	Comparator with 3-point response

INTK Integrator (extended)



The block generates the integral of the controlled variable X multiplied by the proportional Coefficient KP.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Serie90_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		INTK	Instance name
X	Input	INT	Controlled variable
T1_TZ	Input	INT	Integration time scaled to the cycle time
RES	Input	INT	Reset output Y to 0
STOP	Input	BOOL	Integrator stop
SET	Input	BOOL	Set output to INIT value
INIT	Input	INT	Initial value
KP	Input	INT	Proportion coefficient, output as a percentage value
OG	Input	INT	High limit for the manipulated variable Y
UG	Input	INT	Low limit for the manipulated variable Y
OG_MELD	Output	BOOL	Output Y has reached top limit
UG_MELD	Output	BOOL	Output Y has reached low limit
Y	Output	INT	Manipulated variable

Description

The block generates the integral of the controlled variable X multiplied by the proportional Coefficient KP. The integrator output Y can be manipulated as follows:

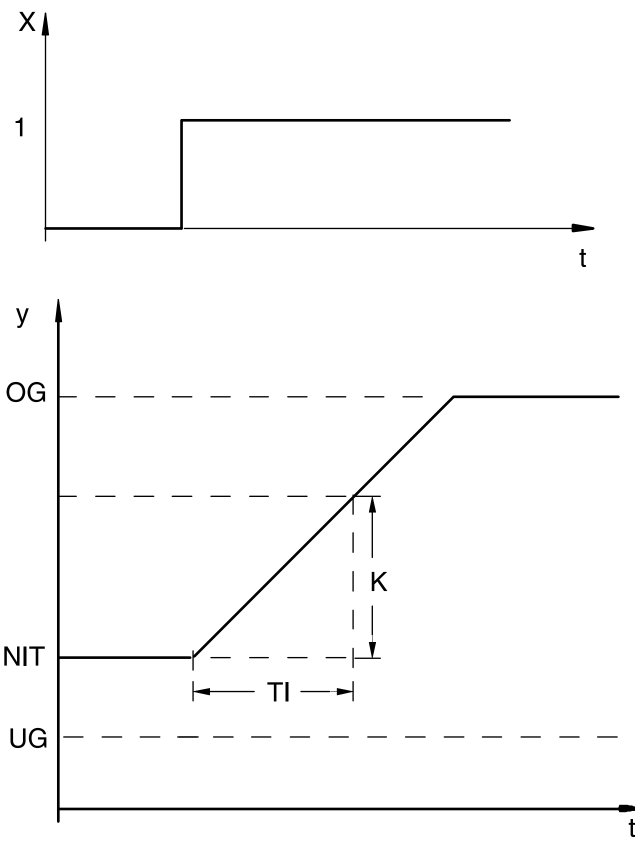
- It can be set to the value 0 by a 1 signal at input RES (reset).
- It can be latched to a current value by a TRUE signal at input STOP.
- It can be set to the initial value at input INIT by a TRUE signal at input SET.
- It can be limited to a maximum value specified at input OG (high limit).
- It can be limited to a minimum value specified at input UG (low limit).

The inputs and outputs can neither be duplicated nor negated/inverted.

Transfer function

$$F(s) = \frac{KP}{s * T}$$

Transient function:



X INT

The operand for the controlled variable is specified at input X.

T1_TZ INT

The integration time is specified at input T1_TZ. In this case, it must be scaled to the cycle time. During the time T1, the output Y of the integrator changes by the value KP * X.

Value range: $0 \le T1_TZ \le 328$

- If values are specified which are beyond the admissible value range, the PLC generally uses the value 328.

- A large integration time (T1) can be achieved by choosing a large cycle time, too. If the block is used within a run number block, the cycle time of the run number block is valid for INTK and not the cycle time of the PLC program.

RES BOOL *)

The output Y can be reset to the value 0 with the input RES. Integration then begins as from the value 0.

STOP BOOL *)

Integration can be stopped with input STOP.

STOP = FALSE
→ Integration is not stopped.

STOP = TRUE
→ Integration is stopped, i.e. output Y no longer changes.

SET BOOL *)

With the input SET, the manipulated value Y can be set to the initial value at input INIT. Integration then begins as from the initial value.

SET = FALSE
→ No setting

SET = TRUE
→ Output Y is set to the specified initial value.

*) Priority sequence for the inputs STOP, SET and RES:
RES highest priority
STOP
SET lowest priority

INIT INT

The initial value to which output Y is to be set is specified at input INIT if required.

KP INT

The proportional coefficient is specified at input KP. It serves to weight the controlled variable at input X. Weighting is achieved by multiplying the controlled variable by the proportional coefficient. The proportional coefficient is specified as a percentage.

Example:

KP	is equal to	Meaning
1	1 percent	The block multiplies the value at input X by the factor 0.01
55	55 percent	The block multiplies the value at input X by the factor 0.55
100	100 percent	The block multiplies the value at input X by the factor 1
1000	1000 percent	The block multiplies the value at input X by the factor 10
-100	-100 percent	The block multiplies the value at input X by the factor -1

OG INT

The manipulated variable Y can be limited to a value range. The high limit for the manipulated variable Y is specified at input OG.

UG INT

The manipulated variable Y can be limited to a value range. The low limit for the manipulated variable Y is specified at input UG.

OG_MELD BOOL

Whether the value at output Y has reached the specified high limit is signalized at output OG_MELD. Integration is stopped automatically when the limit is reached.

OG_MELD = FALSE
→ Output Y has not reached the limit (yet).

OG_MELD = TRUE
→ Output Y has reached the limit.

UG_MELD BOOL

Whether the value at output Y has reached the specified low limit is signalized at output UG_MELD. Integration is stopped automatically when the limit is reached.

UG_MELD = FALSE
→ Output Y has not reached the limit (yet).

UG_MELD = TRUE
→ Output Y has reached the limit.

Y INT

The manipulated variable (output value of the integrator) is provided at the output Y.

Function call in IL

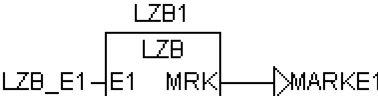
```
CAL   INTK1(X := INTK_X, T1_TZ := INTK_T1TZ,
      RES := INTK_RES, STOP := INTK_STOP,
      SET := INTK_SET, INIT := INTK_INIT,
      KP := INTK_KP, OG := INTK_OG,
      UG := INTK_UG)

LD    INTK1.OG_MELD
ST    INTK_MOG

LD    INTK1.UG_MELD
ST    INTK_MUG

LD    INTK1.Y
ST    INTK_Y
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
INTK1(X := INTK_X, T1_TZ := INTK_T1TZ,
RES := INTK_RES, STOP := INTK_STOP,
SET := INTK_SET, INIT := INTK_INIT,
KP := INTK_KP, OG := INTK_OG, UG := INTK_UG);
INTK_MOG := INTK1.OG_MELD;
INTK_MUG := INTK1.UG_MELD;
INTK_Y := INTK1.Y;
```

LZB Run number block



The block controls processing of a program part. This program part is called run number block and begins with the function block LZB and ends with the appropriate target label specified at the block output MRK. This program part is processed as follows depending on the value of the operand at input E1.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Serie90_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		LZB	Instance name
E1	Input	INT	Run number
MRK	Output	SPRUNG	Target label

Description

The block controls processing of a program part. This program part is called run number block and begins with the function block LZB and ends with the appropriate target label specified at the block output MRK. This program part is processed as follows depending on the value of the operand at input E1:

E1 = 0:	Program part is not processed.
E1 = 1:	Program part is processed during every cycle.
E1 = 2:	Program part is processed during every second cycle.
:	:
E1 = n:	Program part is processed during every nth cycle.

E1 INT

This program part is processed as follows depending on the value of the operand at input E1:

E1 = 0:	Program part is not processed.
E1 = 1:	Program part is processed during every cycle.
E1 = 2:	Program part is processed during every second cycle.
:	:
E1 = n:	Program part is processed during every nth cycle.

MRK BOOL

At this output, a jump instruction with a corresponding jump destination must be specified. Output MRK only signalizes, whether the subsequent program part is processed or not.

The following applies:

MRK = FALSE
→ Processing of program part

MRK = TRUE
→ No processing of program part

Function call in IL

```
CAL  LZB1(E1 := LZB_E1)

LD   LZB1.MRK
JMPC MARKE
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
LZB(E1 := LZB_E1);

IF (LZB1.MRK)
     THEN .....;
END_IF
```

VGL3P Comparator with 3-point response

```
              VGL3P1
            +---------+
            |  VGL3P  |
  VGL3_E ---|E    E_OG|--------------VGL3_EOG
 VGL3_OG ---|OG   E_UG|---VGL3_EUG
 VGL3_UG ---|UG      Q|---VGL3_Q
            +---------+
```

The value of the operand at input E is compared to the values of the operands at the inputs OG and UG.

The possible results are signalled at the outputs E_OG, E_UG and Q.

The following applies:

E < UG
→ E_OG = FALSE, E_UG = TRUE, Q = FALSE

UG ≤ E ≤ OG
→ E_OG FALSE, E_UG = FALSE, Q = TRUE

E > OG
→ E_OG TRUE, E_UG = FALSE, Q = FALSE

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Serie90_AC500_V10.LIB	

Block type

Function block without historical values

Parameter

Instance		VGL3P	Instance name
E	Input	INT	Input value
OG	Input	INT	High limit
UG	Input	INT	Low limit
E_OG	Output	BOOL	Value > high limit
E_UG	Output	BOOL	Value < low limit
Q	Output	BOOL	Low limit ≤ input value ≤ high limit

Description

The value of the operand at input E is compared to the values of the operands at the inputs OG and UG.

The possible results are signalled at the outputs E_OG, E_UG and Q.

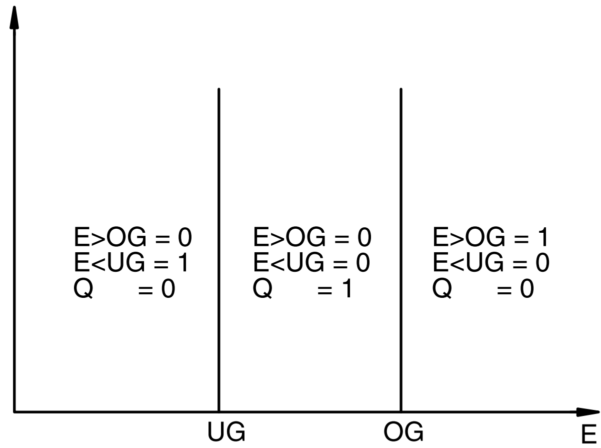
The following applies:

E < UG
→ E_OG = FALSE, E_UG = TRUE, Q = FALSE

UG ≤ E ≤ OG
→ E_OG FALSE, E_UG = FALSE, Q = TRUE

E > OG
→ E_OG TRUE, E_UG = FALSE, Q = FALSE

The inputs and outputs can neither be duplicated nor negated/inverted.



Function call in IL

```
CAL   VGL3P1(E := V3P_E, OG := V3P_OG,
      UG := V3P=_UG)

LD    VGL3P1.E_UG
ST    V3P_EUG

LD    VGL3P1.Q
ST    V3P_Q

LD    VGL3P1.E_OG
ST    V3P_EOG
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
VGL3P1(E := V3P_E, OG := V3P_OG,
UG := V3P_UG);
V3P_EUG := VGL3P1.E_UG;
V3P_Q := VGL3P1.Q
V3P_EOG := VGL3P1.E_OG;
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

Components of the Series90 AC500 Library 2

G

Glossary 11

I

INTK Integrator (extended) 3

L

LZB Run number block 7

V

VGL3P Comparator with 3-point response 9

Software Description

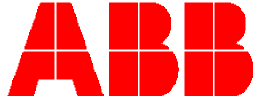
AC500

Scalable PLC
for Individual Automation

ASCII Communication
Function Block Library

ASCII

Communication



Contents

ASCII Communication Library 2

Preconditions for the use of the library 2

Components of the ASCII Communication Library 2

Overview of blocks arranged according to their call names 2

COM_REC Receiving data via a serial interface in the "free mode" 3

COM_SEND Sending data via a serial interface in the "free mode" 7

Glossary 11

Index 13

ASCII Communication Library

Preconditions for the use of the library

Note:

The blocks for ASCII communication can only be executed in RUN mode of the PLC, but not in simulation mode.

Components of the ASCII Communication Library

The ASCII Communication Library contains the following function blocks:

Group: ASCII communication	
COM_REC	Receiving data via a serial interface in the "free mode"
COM_SEND	Sending data via a serial interface in the "free mode"

Setting of the communication interfaces COMx

There is no particular block available for setting the communication interfaces in transmitting and receiving direction. This is done in the AC500 controller configuration within CoDeSys.

Overview of blocks according to their call names

Used abbreviations:

FBhv . .. Function block with historical values

FBnohv ... Function block without historical values

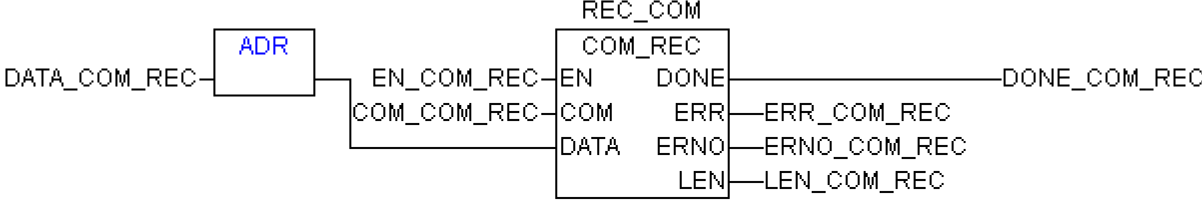
F … Function

VE Name	**Type**	**Function**
COM_REC	FBhv	Receiving data via a serial interface in the "free mode"
COM_SEND	FBhv	Sending data via a serial interface in the "free mode"

Setting of the communication interfaces COMx

There is no particular block available for setting the communication interfaces in transmitting and receiving direction. This is done in the AC500 controller configuration within CoDeSys.

COM_REC Receiving data via a serial interface in the "free mode"



The block COM_REC is used for receiving data via a serial interface in "free mode". The number of COM_REC blocks within a project as well as their distribution (i.e. assignment) to different user tasks is not restricted. However, it has to be observed that the blocks are mutually interlocked, i.e. it must be ensured that only one block is active at the same time. To avoid the loss of telegram parts and to prevent that telegrams are evaluated incorrectly or not at all, a change of activity between two COM_REC blocks should only occur if the block to be deactivated has signalized the termination of the receive process by setting DONE = TRUE and if the received telegram has been evaluated. It is essential to ensure that all block instances are inactive prior to initiating an activity change. Thus, it is strongly recommended to use only one COM_REC block within a project. This way, any responsibility conflicts can be avoided.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	ASCII_AC500_V10.lib	

Block type

Function block with historical values

Parameters

Instance		COM_REC	Instance name
EN	Input	BOOL	Enabling sending by FALSE/TRUE edge
COM	Input	BYTE	Interface number (COM1 or COM2)
DATA	Input	DWORD	Memory address for received data via ADR operator
DONE	Output	BOOL	Data received
ERR	Output	BOOL	Error during data reception
ERNO	Output	WORD	Error number
LEN	Output	WORD	Length of valid received data (in byte) starting at address DATA

Description

The block COM_REC is used for receiving data via a serial interface in "free mode". The number of COM_REC blocks within a project as well as their distribution (i.e. assignment) to different user tasks is not restricted. However, it has to be observed that the blocks are mutually interlocked, i.e. it must be ensured that only one block is active at the same time. To avoid the loss of telegram parts and to prevent that telegrams are evaluated incorrectly or not at all, a change of activity between two COM_REC blocks should only occur if the block to be deactivated has signalized the termination of the receive process by setting DONE = TRUE and if the received telegram has been evaluated. It is essential to ensure that all block instances are inactive prior to initiating an activity change. Thus, it is strongly recommended to use only one COM_REC block within a project. This way, any responsibility conflicts can be avoided.

With a FALSE > TRUE edge at block input EN, the block checks the input values. If they are valid, the block reads the receiving buffer of the corresponding COMx interface one-time.

By specifying the length of the memory address for the received data, the format of the telegram is not restricted in any way. The length of the received data block is limited to a maximum of 256 bytes and is output at LEN.

Note: During project planning it has to be observed that enough free memory space is reserved starting at address DATA for storing the received data (e.g. ARRAY [1..256] OF BYTE).

If a valid received telegram is available in the memory area starting at DATA, this is always indicated by DONE = TRUE.

The inputs can neither be duplicated nor negated/inverted.

Receive error

Possible receive errors are detected by the block and indicated by ERR = TRUE. In this case, an error number is output at ERNO. The block recognizes overflow, parity and framing errors. In this case, the communication parameters (baud rate, char length, no. of stop bits, parity) of the communication partners have to be checked.

Times

There is no particular block available for setting the times in transmitting and receiving direction. This is done in the AC500 controller configuration within CoDeSys.

EN BOOL

With a FALSE > TRUE edge at block input EN, the block checks the input values. If they are valid, the block reads the receiving buffer of the corresponding COMx interface one-time.
EN = TRUE – Reading of the receiving buffer
EN = FALSE – No reading of the receiving buffer

COM BYTE

At the COM input, the number of the serial interface is specified.

COM = 1: COM1
COM = 2: COM2

DATA DWORD

At input DATA, the start address for storing the received data is specified using an ADR operator. Received data can be stored in the operand area as well as in variables. Some peculiarities have to be observed when receiving binary values.

When using IEC bit operands as storage address, only operands are allowed which end with ".0" (e.g. %QX62.0 allowed, %QX62.1 forbidden).

When storing a received telegram within the IEC bit operand area, it has to be observed that a received byte describes 8 bit operands. However, if a received telegram is stored to a Boolean variable, this variable is considered as FALSE if the byte has the value 0 and TRUE for all other values of the byte.

DONE BOOL

Output DONE indicates whether a telegram has been received since the last execution of the COM_REC block (DONE = TRUE) or not (DONE = FALSE). DONE has always to be considered in conjunction with output ERR, since errors may have occurred during reception (e.g. buffer too small for actual telegram length).

ERR BOOL

Output ERR indicates whether an error occurred. This output always has to be considered together with output DONE. If an error has occurred, the following applies: DONE = TRUE and ERR = TRUE. Output ERNO signalizes the error number.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

LEN WORD

LEN outputs the length of the received data (in bytes) including end character (if contained). LEN is only valid if DONE = TRUE. LEN has always to be considered together with output ERR.

Function call in IL

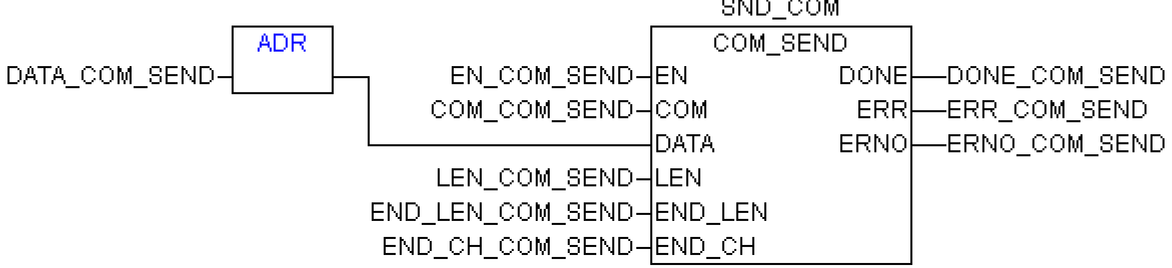
```
LD   DATA_COM_REC
ADR
ST   REC_COM.DATA

CAL  REC_COM
(EN  := EN_COM_REC,
COM  := COM_COM_REC)

LD   REC_COM.DONE
ST   DONE_COM_REC

LD   REC_COM.ERR
ST   ERR_COM_REC

LD   REC_COM.ERNO
ST   ERNO_COM_REC

LD   REC_COM.LEN
ST   LEN_COM_REC
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
REC_COM
(EN := EN_COM_REC,
COM := COM_COM_REC,
DATA := ADR(DATA_COM_REC));
DONE_COM_REC := REC_COM.DONE;
ERR_COM_REC := REC_COM.ERR;
ERNO_COM_REC := REC_COM.ERNO;
LEN_COM_REC := REC_COM.LEN;
```

COM_SEND Sending data via a serial interface in the "free mode"



The function block COM_SEND is used for sending data via a serial interface. The number of COM_SEND blocks in a project as well as their usage in (i.e. assignment to) different user tasks is not restricted. Transmission is triggered by a FALSE > TRUE edge at input EN.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	ASCII_AC500_V10.lib	
	V1.2	Swapping END_CH and Checksum Option added

Block type

Function block with historical values

Parameters

Instance		COM_SEND	Instance name
EN	Input	BOOL	Enabling sending by FALSE/TRUE edge
COM	Input	BYTE	Interface number (COM1 or COM2)
DATA	Input	DWORD	Memory address for data to be transmitted via ADR operator
LEN	Input	WORD	Length of data to be transmitted (in byte) starting at address DATA
END_LEN	Input	BYTE	Number of end characters to be attached (0, 1, 2)
END_CH	Input	WORD	Telegram end character, consisting of a max. of 2 characters (e.g. 16#0D0A) End characters to be attached to the data
DONE	Output	BOOL	Transmit buffer empty
ERR	Output	BOOL	Error during transmission
ERNO	Output	WORD	Error number

Description

The function block COM_SEND is used for sending data via a serial interface. The number of COM_SEND blocks in a project as well as their usage in (i.e. assignment to) different user tasks is not restricted. Transmission is triggered by a FALSE > TRUE edge at input EN. By specifying the length of the memory address for the data to be transmitted, the format of the telegram is not restricted in any way.

The length of the data block to be transmitted is not limited. It is recommended to write only data blocks up to a maximal size of 256 bytes into the transmit buffer, since data can only be transmitted if enough free memory space is available for the transmit buffer. If required, longer telegrams can be generated by using several COM_SEND blocks following each other immediately without considering their individual DONE outputs.

The inputs can neither be duplicated nor negated/inverted.

Telegram length

The maximum length of a transmit telegram can be controlled by evaluating the output DONE. DONE = TRUE indicates an empty transmit buffer. If a transmission is only triggered if DONE = TRUE, one single telegram will be sent per COM_SEND. Theoretically, endless data streams can be generated by ignoring DONE.

EN BOOL

EN FALSE >TRUE

When a FALSE > TRUE edge is detected at input EN, the block tries to transmit the telegram specified at the block inputs. If the check of the input parameters results in an error, this error will be displayed at output ERNO. DONE and ERR are then set to TRUE. In case of an error, no transmission is performed.

COM BYTE

At the COM input, the number of the serial interface is specified.

COM = 1: COM1
COM = 2: COM2

DATA DWORD

At input DATA the start address of the data to be transmitted is specified using an ADR operator. Transmit data can be operand values and variable values. Some peculiarities have to be observed when transmitting binary values.

When using IEC bit operands as telegram start address, only operands are allowed which end with ".0" (e.g. %QX62.0 allowed, %QX62.1 forbidden). If necessary, this restriction can be compassed by copying the IEC bit operands into variables prior to transmitting and specifying the telegram start address as first variable.

IEC bit operands are transmitted byte-wise packed. This can be compassed by splitting the IEC bit operands into Boolean variables prior to transmitting them (e.g. ARRAY [..] OF BOOL). In this case, the first variable of the array has to be specified as telegram start address with the ADR operator.

Using a Boolean variable as telegram start address is possible without restrictions. In contrast to IEC bit operands, Boolean variables are transmitted as 1 bit per byte (value "0" or "1"). If this is not desired, the Boolean variables must be summarized byte-wise prior to transmitting (e.g. using the block PACK) and the packed data must then be copied into the data area to be transmitted (e.g. ARRAY [..] OF BYTE). In this case, the first variable of the array has to be specified as telegram start address with the ADR operator.

<table>
<tr><td rowspan="2">Address</td><td colspan="2">Addr</td><td colspan="2">Addr+1</td><td colspan="2">Addr+2</td><td colspan="2">Addr+3</td></tr>
<tr><td colspan="2">16#0041 0000</td><td colspan="2">16#0041 0001</td><td colspan="2">16#0041 0002</td><td colspan="2">16#0041 0003</td></tr>
<tr><td rowspan="2">BYTE</td><td colspan="2">%IB0</td><td colspan="2">%IB1</td><td colspan="2">%IB2</td><td colspan="2">%IB3</td></tr>
<tr><td>7</td><td>0</td><td>7</td><td>0</td><td>7</td><td>0</td><td>7</td><td>0</td></tr>
<tr><td>BOOL</td><td>%IX0.7</td><td>%IX0.0</td><td>%IX1.7</td><td>%IX1.0</td><td>%IX2.7</td><td>%IX2.0</td><td>%IX3.7</td><td>%IX3.0</td></tr>
<tr><td rowspan="2">WORD</td><td colspan="4">%IW0</td><td colspan="4">%IW1</td></tr>
<tr><td>15</td><td>8</td><td>7</td><td>0</td><td>15</td><td>8</td><td>7</td><td>0</td></tr>
<tr><td rowspan="2">DWORD</td><td colspan="8">%ID0</td></tr>
<tr><td>31</td><td>24</td><td>23</td><td>16</td><td>15</td><td>8</td><td>7</td><td>0</td></tr>
</table>

Example: Addressing in BOOL / BYTE / WORD / DWORD

LEN WORD

At input LEN the length of the data to be transmitted (in bytes) is specified without possible end character. LEN is applied with a FALSE > TRUE edge at EN.

END_LEN BYTE

Number of end characters to be attached: 0, 1, 2 means none, one or two.

END_CH WORD

At input END_CH, the value of the telegram end character(s) must be specified which has (have) to be attached to the actual transmit data. END_CH is considered if input END_LEN of the COM_SEND block is not 0. END_CH is applied with a FALSE > TRUE edge at EN. If required, the end character can vary within each telegram.

If, for example, a CR/LF shall be attached to a telegram, the value of END_CH is as follows: 16#0D0A and END_LEN: 2.

DONE BOOL

Block output DONE indicates whether the sending buffer is completely empty (DONE = TRUE) or not (DONE = FALSE). This is independent from the state of input EN. Precondition is that the interface has already been configured successfully as "free" interface and is active accordingly. DONE has always to be considered together with ERR. If ERR = TRUE, an error number is available at output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred. This output always has to be considered together with output DONE. If an error has occurred, the following applies: DONE = TRUE and ERR = TRUE. Output ERNO signalizes the error number.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO is explained at the beginning of the block description.

Function call in IL

```
LD   DATA_COM_SEND
ADR
ST   SEND_COM.DATA

CAL      SEND_COM
(EN      := EN_COM_SEND,
COM      := COM_COM_SEND,
LEN      := LEN_COM_SEND,
END_CH   := END_CH_COM_SEND,
END_LEN := END_LEN_COM_SEND)

LD   SEND_COM.DONE
ST   DONE_COM_SEND

LD   SEND_COM.ERR
ST   ERR_COM_SEND

LD   SEND_COM.ERNO
ST   ERNO_COM_SEND
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
SEND_COM
(EN := EN_COM_SEND,
COM := COM_COM_SEND,
DATA := ADR(DATA_COM_SEND),
LEN := LEN_COM_SEND,
END_CH := END_CH_COM_SEND,
END_LEN := END_LEN_COM_SEND);

DONE_COM_SEND := SEND_COM.DONE;

ERR_COM_SEND := SEND_COM.ERR;
ERNO_COM_SEND := SEND_COM.ERNO;
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

COM_REC Receiving data via a serial interface in the "free mode" 3

COM_SEND Sending data via a serial interface in the "free mode" 7

Components of the ASCII Communication Library 2

G

Glossary 11

O

Overview of blocks arranged according to their call names 2

P

Preconditions for the use of the library 2

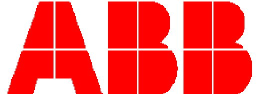
Software Description

AC500

Scalable PLC
for Individual Automation

CS31
Function Block Library

CS31



Contents

CS31 Library, function blocks for the CS31 bus 2

Preconditions for the use of the CS31 library 2

Components of the CS31 library 2

Overview of blocks arranged according to their call names 2

CS31CO Configure AC31 modules 3

CS31QU Acknowledge AC31 errors 14

Glossary 16

Index 18

CS31 Library

Preconditions for the use of the CS31 Library

Note:

The blocks of the CS31 Library can only be executed in RUN mode of the PLC, but not in simulation mode.

⚠ **Important:** The function blocks of the CS31 library can only be called, if the serial interface COM1 is configured as "CS31 bus master".

Components of the CS31 Library

The CS31 Library contains the following function blocks:

CS31CO Configure AC31 modules

CS31QU Acknowledge AC31 errors

Overview of blocks arranged according to their call names

Used abbreviations:

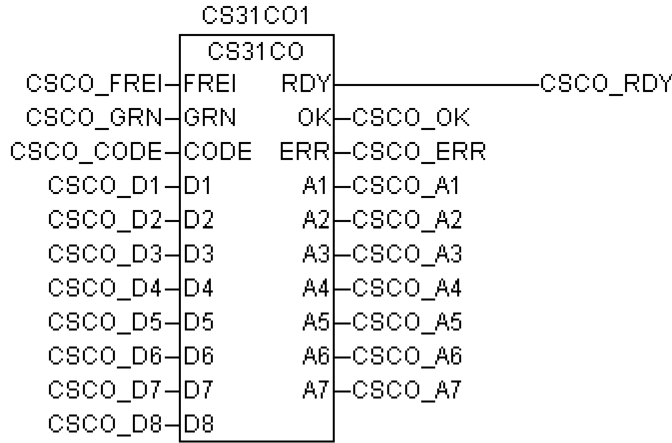
FBhv ... Function block with historical values

FBnohv ... Function block without historical values

F … Function

VE name	Type	Function
CS31CO	FBhV	Configure AC31 modules
CS31QU	FBhV	Acknowledge AC31 errors

CS31CO Configure AC31 modules



The function block is used to configure the AC31 remote modules. The block can both send configuration parameters to the remote modules and also scan their currently set configuration.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	CS31_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		CS31CO	Instance name
FREI	Input	BOOL	Enable (FALSE/TRUE edge) for processing the block
GRN	Input	INT	Group number of the remote module to which the job refers
CODE	Input	INT	Identification of the job to be performed
D1	Input	INT	1st parameter of the job
.	.	.	.
.	.	.	.
D8	Input	INT	8th parameter of the job
RDY	Output	BOOL	Processing of the job is completed
OK	Output	BOOL	The job could be processed correctly
ERR	Output	INT	Error message/status message
A1	Output	INT	1st parameter of the response
.	.	.	.
.	.	.	.
A7	Output	INT	7th parameter of the response

Description

The function block is used to configure the AC31 remote modules. The block can both send configuration parameters to the remote modules and also scan their currently set configuration.

Apart from configuration of the AC31 remote modules, the function block can also process further jobs (see list of jobs).

Enable for processing a job once is triggered by a FALSE/TRUE edge at input FREI.

The required job identification is specified at input CODE.

The parameters required for the job are planned at the inputs D1 ... D8.

Status messages are signalized at the outputs RDY, OK and ERR.

The response data of the job are available at the outputs A1 ... A7.

It may take several PLC cycles to process the job.

FREI BOOL

Processing of the block is controlled via input FREI.

FREI = FALSE:
All block outputs are set to the value "FALSE". However, this is not valid, if a job is currently being processed, i. e. processing of a job which is currently being processed, is not affected by FREI = FALSE.

FREI = FALSE/TRUE edge:
Processing of the job is enabled. Input FREI is no longer evaluated during processing of the job.

FREI = TRUE:
The block is not processed, i. e. it no longer changes its outputs. However, this is not valid, if a job is currently being processed.

GRN INT

Group number with which the remote module is addressed by the PLC program.

Range: 0...63

Example:
On binary input E 12,08, "12" is the group number and "08" is the channel number.

CODE INT

The identification of the job to be executed is specified at input CODE (see list of jobs).

D1...D8 INT

The parameters required for the job are preset at the inputs D1 ... D8. The number of parameters depends on the job to be executed. There are also jobs requiring no parameters (see list of jobs).

RDY BOOL

The output RDY indicates that processing of the job currently being processed is completed. This output does not indicate whether processing of the job was successful or not. The output RDY has therefore always to be considered together with the output OK.

RDY = TRUE and OK = TRUE:
Processing of the job is completed without errors. A new job can be started with a FALSE/TRUE edge at input FREI.

RDY = TRUE and OK = FALSE:
During processing of the job an error has been detected. The corresponding error identification is present at output ERR. A new job can be started with a FALSE/TRUE edge at input FREI.

RDY = FALSE
Processing of an enabled job has not yet been completed (job is still running) or output RDY has been reset with FREI = FALSE.

OK BOOL

Output OK indicates whether the job has been handled successfully or whether an error has been detected during processing. In case of an error, an error number is indicated at output ERR. The output OK is not valid until the job has been completed, i. e. if RDY = TRUE.

The following applies:
If RDY = TRUE and

OK = TRUE: The job has been processed successfully.

OK = FALSE: During processing of the job an error has been detected.

ERR INT

At the output ERR status and error identifications are output. The status identifications are output during processing of a job in order to signalize in what stage of processing the job currently is. After enabling a job, status identifications are signalized only for as long as RDY = FALSE.

The error identifications are output after completion of the job processing if an error has occurred. Error identifications are thus not signalized until

RDY = TRUE and
OK = FALSE

Error identifications

ERR = 1: An illegal job identification has been specified at input CODE.

ERR = 2: Incorrect parameters have been specified at the inputs D1...D8 (e.g. a group number for which there is no remote module on the CS31 system bus).

ERR = 3: The addressed AC31 remote module does not accept the job.

Status identifications

ERR = 8: The function block is waiting since a job of another user is currently being processed.

ERR = 10: The job has been sent to the receiver and the block is waiting for its response.

A1...A7 INT

After completion of job processing, the response is available at the outputs A1 ... A7. The number of response parameters depends on the job performed (see list of jobs).

List of jobs

Processing a job consists of:
- transferring the job and
- supplying the OK response or not-OK response

The OK response is described in connection with the corresponding job.

The not-OK response of the individual jobs always looks as follows:

** Not-OK response*

The following basically applies for the not-OK response:

RDY: TRUE
OK: FALSE

ERR: 1. inadmissible job identification
2. wrong parameter; e. g. group number to which there exists no remote module
3. remote module does not accept the job

A1...A7: 0

Updating of the maximum number of remote modules detected

The input word EW 07,15 contains, amongst other things, the maximum number of remote modules detected in the past. The actual number of remote modules which exist at the moment may be less. This command is used to update this value. The modules which exist are counted and the value is stored. The user can inquire this value in the PLC program (EW 07,15, bit 8...15).

-	Job GRN: CODE: D1...D8:	 255 (Master PLC with bus) 132 Not used
-	OK response RDY: OK: A1...A7:	 TRUE TRUE 0

Inquiring the open-circuit monitoring of an input to determine whether it is activated or deactivated

-	Job GRN: CODE: D1: D2...D8:	 Group number 0...63 32 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	 TRUE TRUE 47. Open-circuit monitoring ON 32. Open-circuit monitoring OFF 0

Inquiring the open-circuit monitoring of an output to determine whether it is activated or deactivated

-	Job GRN: CODE: D1: D2...D8:	 Group number 0...63 33 Channel number Not used
-	OK response RDY: OK:	 TRUE TRUE

	A1: A2...A7:	47. Open-circuit monitoring ON 32. Open-circuit monitoring OFF 0

Deactivating or activating the open-circuit monitoring of an input

-	Job GRN: CODE: D1: D2...D8:	 Group number 0...63 224. Open-circuit monitoring ON 160. Open-circuit monitoring OFF Channel number Not used
-	OK response RDY: OK: A1...A7:	 TRUE TRUE 0

Deactivating or activating the open-circuit monitoring of an output

-	Job GRN: CODE: D1: D2...D8:	 Group number 0...63 225. Open-circuit monitoring ON 161. Open-circuit monitoring OFF Channel number Not used
-	OK response RDY: OK: A1...A7:	 TRUE TRUE 0

Inquiring a channel to determine whether it is configured as input or input/output

-	Job GRN: CODE: D1: D2...D8:	 Group number 0...63 34 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	 TRUE TRUE 34. Input 35. Input/output 0

Configuration of a channel as input or input/output

-	Job GRN: CODE: D1: D2...D8:	 Group number 0...63 162. Input 163. Input/output Channel number Not used
-	OK response RDY: OK: A1...A7:	 TRUE TRUE 0

Inquiring the input delay of a channel

-	Job GRN: CODE: D1: D2...D8:	 Group number 0...63 38 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	 TRUE TRUE Input delay 2. 2 ms 4. 4 ms : : 30. 30 ms 32. 32 ms 0

Setting the input delay of a channel

-	Job GRN: CODE: D1: D2:	 Group number 0...63 166 Channel number Input delay 2. 2 ms 4. 4 ms : : 30. 30 ms 32. 32 ms
-	OK response RDY: OK: A1...A7:	 TRUE TRUE 0

Acknowledging errors on remote module

This command can be used to reset the error messages registered on the selected remote module. A reset is possible only if the cause of the error is no longer operative.

-	Job GRN: CODE: D1: D2: D3...D8:	 Group number 0...63 232 Lowest channel number on the module: 0. Lowest channel number on the module is 0 (<7) 8. Lowest channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Wort: odd number (1, 3, 5) Not used
-	OK response RDY: OK: A1...A7:	 TRUE TRUE 0

Acknowledging errors on remote module and resetting configuration values to default setting

In addition to the job "Acknowledging errors on remote module", all configurable settings are reset to the default setting.

-	Job	
	GRN:	Group number 0...63
	CODE:	233
	D1:	First channel number on the module:
		0. First channel number on the module is 0 (<7)
		8. First channel number on the module is 8 (>7)
	D2:	Module type:
		0. Binary input
		1. Analog input
		2. Binary output
		3. Analog output
		4. Binary input/output
		5. Analog input/output
		Note:
		Bit: even number (0, 2, 4)
		Wort: odd number (1, 3, 5)
	D3...D8:	Not used
-	OK response	
	RDY:	TRUE
	OK:	TRUE
	A1...A7:	0

Inquiring the configuration of an analog input

-	Job	
	GRN:	Group number 0...63
	CODE:	42
	D1:	Channel number
	D2...D8:	Not used
-	OK response	
	RDY:	TRUE
	OK:	TRUE
	A1:	50. Input 0...20 mA
		49. Input 4...20 mA
	A2...A7:	0

Inquiring the configuration of an analog output

-	Job	
	GRN:	Group number 0...63
	CODE:	43
	D1:	Channel number
	D2...D8:	Not used
-	OK response	
	RDY:	TRUE
	OK:	TRUE
	A1:	50. Output 0...20 mA
		49. Output 4...20 mA
	A2...A7:	51. Output +10 V
		0

Configuration of an analog input

-	Job	
	GRN:	Group number 0...63
	CODE:	170
	D1:	Channel number
	D2:	50. Input 0...20 mA
		49. Input 4...20 mA
	D3...D8:	Not used
-	OK response	
	RDY:	TRUE
	OK:	TRUE
	A1...A7:	0

Configuration of an analog output

-	Job	
	GRN:	Group number 0...63
	CODE:	171
	D1:	Channel number
	D2:	50. Output 0...20 mA
		49. Output 4...20 mA
		51. Output +10 V
	D3...D8:	Not used
-	OK response	
	RDY:	TRUE
	OK:	TRUE
	A1...A7:	0

Inquiring the bus configuration

The bus interface of the Master PLC has a list which stores specific data of the remote modules. The remote modules are numbered in this list in the order in which they can be found on the CS31 system bus. The internal number of the modules must be specified with this command. The response to this command is the group number stored under this number and status information on the corresponding module.

-	Job	
	GRN:	Not evaluated
	CODE:	80
	D1:	Number from the module list (1...31)
	D2...D8:	Not used
-	OK response	
	RDY:	TRUE
	OK:	TRUE
	A1:	Status of the remote module:
		Bits 0...3: Number of process data bytes (binary module) or words (word module), which the module sends to the master.
		Bits 4...7: Number of process data bytes (binary module) or words (word module), which the master sends to the module
	A2:	Group number
	A3:	Bit 0: 0. Lowest channel number <7
		1. Lowest channel number >7
		Bit 1: 0. Binary module
		1. Word module
	A4...A7:	0

Read 1 ... 6 bytes

-	Job GRN: CODE: D1: D2: D3: D4: D5...D8:	 Group number 0...63 49. Read 1 byte 50. Read 2 bytes 51. Read 3 bytes 52. Read 4 bytes 53. Read 5 bytes 54. Read 6 bytes First channel number on the module: 0. First channel number on the module is 0 (<7) 1. First channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5) Byte start address (Low Byte) Byte start address (High Byte) Not used
-	OK response RDY: OK: A1: A2: A3: A4. A5: A6: A7:	 TRUE TRUE Value of the 1st byte Value of the 2nd byte or 0 Value of the 3rd byte or 0 Value of the 4th byte or 0 Value of the 5th byte or 0 Value of the 6th byte or 0 0

Read 1 bit from 1 byte

-	Job GRN: CODE: D1: D2: D3: D4: D5: D6...D8:	 Group number 0...63 63 First channel number of the module: 0. First channel number of the module is 0 (<7) 1. First channel number of the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5) Byte start address (Low Byte) Byte start address (High Byte) Bit position within the byte 0...7 Not used
-	OK response RDY: OK: A1: A2...A7:	 TRUE TRUE Bit value (0 or 1) 0

Write 1...4 bytes

-	Job	
	GRN:	Group number 0...63
	CODE:	65. Write 1 byte 66. Write 2 bytes 67. Write 3 bytes 68. Write 4 bytes
	D1:	First channel number on the module: 0. First channel number on the module is 0 (<7) 1. First channel number on the module is 8 (>7)
	D2:	Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5)
	D3:	Byte start address (Low Byte)
	D4:	Byte start address (High Byte)
	D5:	Value of the 1st byte
	D6:	Value of the 2nd byte or not used
	D7:	Value of the 3rd byte or not used
	D8:	Value of the 4th byte or not used
-	OK response	
	RDY:	TRUE
	OK:	TRUE
	A1...A7:	0

Write 1 bit of 1 byte

-	Job	
	GRN:	Group number 0...63
	CODE:	79
	D1:	First channel number on the module: 0. First channel number on the module is 0 (<7) 1. First channel number on the module is 8 (>7)
	D2:	Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5)
	D3:	Byte start address (Low Byte)
	D4:	Byte start address (High Byte)
	D5:	Bit position within the byte 0...7
	D6:	Bit value (0 or 1)
	D7...D8:	Not used
-	OK response	
	RDY:	TRUE
	OK:	TRUE
	A1...A7:	0

Function call in IL

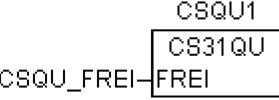
```
CAL  CS31CO1(FREI := CSCO_FREI,
     GRN := CSCO_GRN,
     CODE := CSCO_CODE, D1 := CSCO_D1,
     D2 := CSCO_D2, D3 := CSCO_D3,
     D4 := CSCO_D4, D5 := CSCO_D5,
     D6 := CSCO_D6, D7 := CSCO_D7,
     D8 := CSCO_D8)

LD   CS31CO1.OK
ST   CSCO_OK
LD   CS31CO1.ERR
ST   CSCO_ERR
LD   CS31CO1.A1
ST   CSCO_A1
LD   CS31CO1.A2
ST   CSCO_A2
LD   CS31CO1.A3
ST   CSCO_A3
LD   CS31CO1.A4
ST   CSCO_A4
LD   CS31CO1.A5
ST   CSCO_A5
LD   CS31CO1.A6
ST   CSCO_A6
LD   CS31CO1.A7
ST   CSCO_A7
LD   CS31CO1.RDY
ST   CSCO_RDY
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
CS31CO1(FREI := CSCO_FREI,
GRN := CSCO_GRN,CODE := CSCO_CODE,
D1 := CSCO_D1, D2 := CSCO_D2, D3 := CSCO_D3,
D4 := CSCO_D4, D5 := CSCO_D5, D6 := CSCO_D6,
D7 := CSCO_D7, D8 := CSCO_D8);

CSCO_OK:=CS31CO1.OK;
CSCO_ERR:=CS31CO1.ERR;
CSCO_A1:=CS31CO1.A1;
CSCO_A2:=CS31CO1.A2;
CSCO_A3:=CS31CO1.A3;
CSCO_A4:=CS31CO1.A4;
CSCO_A5:=CS31CO1.A5;
CSCO_A6:=CS31CO1.A6;
CSCO_A7:=CS31CO1.A7;
CSCO_RDY:=CS31CO1.RDY;
```

CS31QU Acknowledge AC31 errors



This block allows to acknowledge automatically error messages of AC31 remote modules.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	CS31_AC500_V10.LIB	

Block type

Function block with historical values

Parameter

Instance		CS31QU	Instance name
FREI	Input	BOOL	Enabling of the block processing

Description

This function block allows to acknowledge automatically error messages of AC31 remote modules. Error messages are stored on the AC31 remote modules until they are acknowledged. Even if the error has been removed, the error message is still pending on the module until acknowledgement and is also signalized to the PLC until the message is acknowledged.

Processing of the block is enabled with a TRUE signal at input FREI, and the block then acknowledges AC31 errors continuously.

It may take several PLC cycles to acknowledge an error on an AC31 module.

If the function block is enabled, it constantly checks whether an AC31 error of class 3 or 4 has occurred and acknowledges this error.

An AC31 error of class 3 has occurred:

The block acknowledges the error on the AC31 remote module which signalizes the error and also clears the error message on the PLC, i.e. the error flag M 255,13 is reset and LED FK3 is deactivated.

Example of a FK3 error:
A remote module is disconnected from the CS31 system bus.

An AC31 error of class 4 has occurred:

The block acknowledges the error on the AC31 remote module which signalizes the error and also clears the error message on the PLC, i.e. the error flag M 255,14 is reset.

Example of a FK4 error:
A remote module signalizes an open circuit.

Function call in IL

```
CAL    CS31QU1(FREI := CSQU_FREI)
```

Function call in ST

```
CS31QU1(FREI := CSQU_FREI);
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

Components of the CS31 Library 2

CS31CO Configure AC31 modules 3

CS31QU Acknowledge AC31 errors 14

G

Glossary 16

O

Overview of blocks arranged according to their call names 2

P

Preconditions for the use of the CS31 Library 2

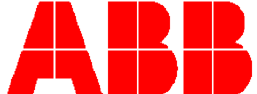
Software Description

AC500

Scalable PLC
for Individual Automation

Ethernet
Function Block Library

Ethernet



Contents

Ethernet Library 2

Preconditions for the use of the library 2

Special characteristics of the Ethernet library 2

Components of the Ethernet library 3

Overview of blocks arranged according to their call names 4

ETH_MOD_INFO Reading status information from the OpenModbus on TCP/IP processing 5

ETH_MOD_MAST Processing OpenModbus on TCP/IP Client (master) telegrams 8

ETH_OWN_IP Outputting the own IP address 12

ETH_UDP_INFO Reading status information from the UDP/IP processing 15

ETH_UDP_REC Reading a data package from the UDP/IP receive buffer 19

ETH_UDP_SEND Sending a data package to a station via Ethernet UDP/IP 23

ETH UDP_STO Reading Ethernet UDP/IP timeout data packages from the timeout data buffer 28

IP_ADR_DWORD_TO_STRING Format conversion of the IP address 31

IP_ADR_STRING_TO_DWORD Format conversion of the IP address 33

Glossary 35

Index 37

Ethernet Library

Preconditions for the use of the Ethernet Library

Note:

The blocks of the Ethernet Library can only be executed in RUN mode of the PLC, but not in simulation mode.

Special characteristics of the Ethernet Library

Note:
Ethernet communication is only performed in RUN mode of the PLC, but not in simulation mode.

Operating the controller as OpenModbus on TCP/IP subscriber can be performed simultaneously with other protocols. When operated in this mode, the Ethernet coupler is able to execute the functionality of several servers or several clients at the same time. Mixed operation is also possible. The corresponding configuration of the coupler has to be performed in SYCON.net (please refer to the SYCON.net documentation).

In order to operate the controller as OpenModbus on TCP/IP server (slave), only the coupler has to be configured accordingly using SYCON.net. An additional use of the OpenModbus on TCP/IP blocks in the user program is not necessary.

To operate the controller as OpenModbus on TCP/IP client (master), the coupler also has to be configured using SYCON.net. In this case, one or more ETH_MODMAST blocks have to be configured in the user program additionally.

The ETH_MODMAST block can be optionally operated in server mode as well as in client mode or in mixed operation.

The following ports are reserved:

Port		Reserved for
DEC	**HEX**	
32768	8000	Ethernet UDP/IP data exchange (ETH_UDP_xxx blocks)
1200	04B0	TCP/IP gateway access
502	01F6	OpenModbus on TCP/IP

Components of the Ethernet Library

The library "Ethernet_AC500_V10.lib" contains the following function blocks:

Group: General		**Page**
ETH_OWN_IP	Outputting the own IP address	12

Group: UDP/IP		**Page**
ETH_UDP_INFO	Reading status information from the UDP/IP processing	15
ETH_UDP_REC	Reading a data package from the UDP/IP receive buffer	19
ETH_UDP_SEND	Sending a data package to a station via Ethernet UDP/IP	23
ETH_UDP_STO	Reading Ethernet UDP/IP timeout data packages from the timeout data buffer	28

Group: MODBUS_TCP (OpenModbus on TCP/IP)		**Page**
ETH_MOD_MAST	Processing OpenModbus on TCP/IP Client (master) telegrams	8
ETH_MOD_INFO	Reading status information from the OpenModbus on TCP/IP processing	5

Group: IP Conversions		**Page**
IP_ADR_DWORD_TO_STRING	Format conversion of the IP address	31
IP_ADR_STRING_TO_DWORD	Format conversion of the IP address	33

Overview of blocks arranged according to their call names

Used abbreviations:

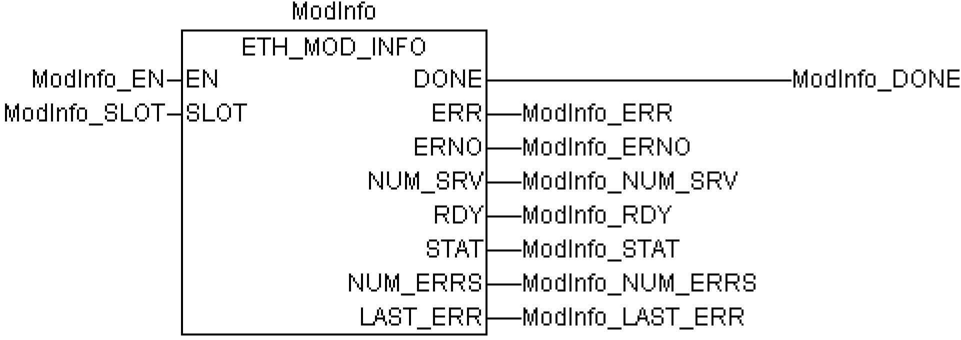
FBhv ... Function block with historical values

FBnohv ... Function block without historical values

F … Function

CE Name	Type	Function	Seite
ETH_MOD_INFO	FBhv	Reading status information from the OpenModbus on TCP/IP processing	5
ETH_MOD_MAST	FBhv	Processing OpenModbus on TCP/IP Client (master) telegrams	8
ETH_OWN_IP	FBhv	Outputting the own IP address	12
ETH_UDP_INFO	FBhv	Reading status information from the UDP/IP processing	15
ETH_UDP_REC	FBhv	Reading a data package from the UDP/IP receive buffer	19
ETH_UDP_SEND	FBhv	Sending a data package to a station via Ethernet UDP/IP	23
ETH_UDP_STO	FBhv	Reading Ethernet UDP/IP timeout data packages from the timeout data buffer	28
IP_ADR_DWORD_TO_STRING	F	Format conversion of the IP address	31
IP_ADR_STRING_TO_DWORD	F	Format conversion of the IP address	33

ETH_MOD_INFO Reading status information from the OpenModbus on TCP/IP processing



The block ETH_MOD_INFO reads the status information of the OpenModbus on TCP/IP processing. It can be used for pure server (slave) or client (master) operation of the controller as well as for mixed operation.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Ethernet_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		ETH_MOD_INFO	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NUM_SRV	Output	BYTE	Number of configured server channels
RDY	Output	BOOL	Ready indication of the OpenModbus processing
STAT	Output	WORD	Status of the OpenModbus processing
NUM_ERRS	Output	DWORD	Number of occurred errors
LAST_ERR	Output	WORD	Identifier of the last occurred error

Description

Using the ETH_MOD_INFO block, various status information about the OpenModbus processing can be read.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs. If the block detects an Ethernet coupler with OpenModbus on TCP/IP functionality in the specified SLOT, this is indicated by DONE = TRUE and ERR = FALSE. The corresponding status information are then available at the block outputs.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

NUM_SRV BYTE (number of servers)

Output NUM_SRV indicates the number of parallel server channels configured with SYCON.net. NUM_SRV is only valid if DONE = TRUE and ERR = FALSE.

RDY BOOL (ready)

RDY indicates the readiness for operation of the OpenModbus on TCP/IP processing. If RDY = TRUE, the server processing as well as the client processing are ready for operation. RDY is only valid if DONE = TRUE and ERR = FALSE.

STAT WORD (status)

Output STAT displays the current operating state of the OpenModubus on TCP/IP processing. STAT is only valid if DONE = TRUE and ERR = FALSE.

STAT		Meaning
DEC	HEX	
0	00	Processing not initialized
1	01	Processing initialized and running
2	02	Processing initialization in progress
3	03	Initialization error
4	04	Processing initialized and waiting for TCP task

NUM_ERRS DWORD (number of errors)

Output NUM_ERRS displays the number of errors that occurred on the Ethernet coupler.

LAST_ERR WORD (last error)

LAST_ERR outputs the last error occurred on the coupler. The error message encoding at output LAST_ERR applies to all Ethernet blocks and is explained at the beginning of the library descriptions.

Function call in IL

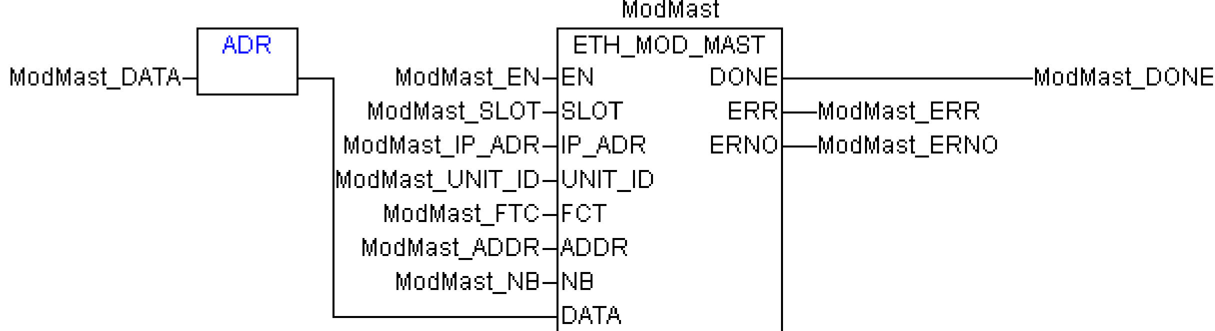
```
CAL  ModInfo(EN := ModInfo_EN, SLOT := ModInfo_SLOT)

LD   ModInfo.DONE
ST   ModInfo_DONE

LD   ModInfo.ERR
ST   ModInfo_ERR

LD   ModInfo.ERNO
ST   ModInfo_ERNO

LD   ModInfo.NUM_SRV
ST   ModInfo_NUM_SRV

LD   ModInfo.RDY
ST   ModInfo_RDY

LD   ModInfo.STATE
ST   ModInfo_STATE

LD   ModInfo.NUM_ERRS
ST   ModInfo_NUM_ERRS

LD   ModInfo.LAST_ERR
ST   ModInfo_LAST_ERR
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
ModInfo (EN := ModInfo_EN,
     SLOT := ModInfo_SLOT);

ModInfo_DONE := ModInfo.DONE;
ModInfo_ERR := ModInfo.ERR;
ModInfo_ERNO := ModInfo.ERNO;
ModInfo_NUM_SRV := ModInfo.NUM_SRV;
ModInfo_RDY := ModInfo.RDY;
ModInfo_STAT := ModInfo.STAT;
ModInfo_NUM_ERRS := ModInfo.NUM_ERRS;
ModInfo_LAST_ERR := ModInfo.LAST_ERR
```

ETH_MOD_MAST Processing OpenModbus on TCP/IP Client (master) telegrams



The ETH_MOD_MAST block can be used to send an OpenModbus on TCP/IP telegram to a server (slave) and to process the corresponding response.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Ethernet_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		ETH_MOD_MAST	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
IP_ADR	Input	DWORD	IP address of the server
UNIT_ID	Input	BYTE	Slave subaddress
FCT	Input	BYTE	MODBUS function code
ADDR	Input	WORD	Operand/register address in the server
NB	Input	WORD	Number of data to be read/written
DATA	Input	DWORD	Address of the first operand in the client from which data shall be written to the server or where the data read from the server shall be stored
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The ETH_MOD_MAST block implements the OpenModbus on TCP/IP client functionality for the Ethernet coupler specified at input SLOT. Depending on the configuration of the coupler, several

ETH_MOD_MAST blocks can be used in parallel. Prior to the use of ETH_MOD_MAST for an Ethernet coupler the coupler has to be configured accordingly using SYCON.net.

With each FALSE > TRUE edge at input EN, the function block ETH_MOD_MAST reads the values at the inputs, generates a telegram according to the inputs and then sends this telegram to the slave.

EN BOOL (enable)

If a FALSE -> TRUE edge is applied to input EN, all further inputs are read in. If the input values are valid, a request telegram is sent to the specified server. If at least one input is invalid, no telegram is generated and the error is displayed at output ERR instead.

While the request is processed, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

IP_ADR DWORD (IP address)

At IP_ADR, the IP address of the server has to be specified to which the telegram should be sent. Each byte in IP_ADR represents one octet of the address.

Example:
IP address 192.15.24.2,
IP_ADR (hex) 16#C00F1802,
IP_ADR (dec) 3222214658

UNIT_ID BYTE (unit ID)

At input UNIT_ID, the address of the MODBUS slave must be specified which is connected serially to the MODBUS server defined by IP_ADR. If no further slaves are connected, this input is not used.

FCT BYTE (function code)

The function code of the request telegram is specified at input FCT. The following function codes are supported:

01 or 02	read n bits
03 or 04	read n words
05	write one bit
06	write one word
07	read M01,00...M01,07
15	write n bits
16	write n words

ADDR WORD (address)

Input ADDR is used to specify the operand/register address in the server from which on data should be read or written. The access to operands of AC500 devices in OpenModbus on TCP/IP (server mode) is defined via the MODBUS address list.

Only operands that are listed in the MODBUS address list may be used. When accessing other devices, ADDR is freely selectable. The valid ranges have to be gathered from the corresponding device description.

NB WORD (number)

At input NB, the number of data to be written or read is specified. The unit of NB depends on the selected function. For bit accesses the number of bits, for word and double word accesses the number of words is specified at NB. The following restrictions apply to the length:

FCT	NBmax
01 or 02	255 bits
03 or 04	100 words / 50 double words
05	1 bit
06	1 word
07	8 bits
15	255 bits
16	100 words / 50 double words

DATA DWORD (data)

At input DATA, the address of the first operand in the client, from which on the data shall be written to the server or where the data read by the server shall be stored, is specified via the ADR operator. For this purpose it is necessary that the operand type (e.g. bit) matches the selected function (e.g. FCT 1, read n bits).

DONE BOOL (done)

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

Function call in IL

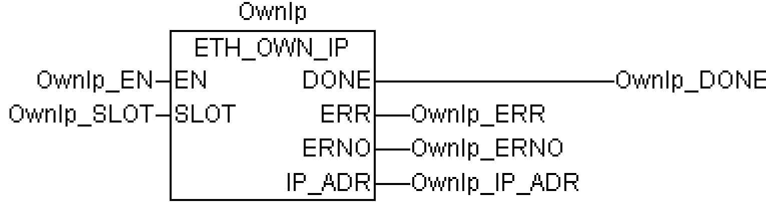
```
LD   ModMast_DATA
ADR
ST   ModMast.DATA

CAL  ModMast
     (EN := ModMast_EN,
     SLOT := ModMast_SLOT,
     IP_ADR := ModMast_IP_ADR,
     UNIT_ID := ModMast_UNIT_ID,
     FCT := ModMast_FCT,
     ADDR := ModMast_ADDR,
     NB := ModMast_NB,
     DATA := ModMast_DATA)

LD   ModMast.DONE
ST   ModMast_DONE

LD   ModMast.ERR
ST   ModMast_ERR

LD   ModMast.ERNO
ST   ModMast_ERNO
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
ModMast (EN := ModMast_EN,
         SLOT := ModMast_SLOT,
         IP_ADR := ModMast_IP_ADR,
         UNIT_ID := ModMast_UNIT_ID,
         FCT := ModMast_FCT,
         ADDR := ModMast_ADDR,
         NB := ModMast_NB,
         DATA := ADR(ModMast_DATA));

ModMast_DONE := ModMast.DONE;
ModMast_ERR := ModMast.ERR;

ModMast_ERNO := ModMast.ERNO;
```

ETH_OWN_IP Outputting the own IP address



The block ETH_OWN_IP outputs the IP address of the coupler installed at slot SLOT.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Ethernet_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		ETH_OWN_IP	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
IP_ADR	Output	DWORD	Own IP address of the coupler

Description

Prior to commissioning a coupler, it has to be configured using the field bus configurator SYCON.net. The IP address is one of the parameters of an Ethernet coupler. Using the block ETH_OWN_IP, the configured IP address of the device at SLOT can be read. If no Ethernet coupler is installed at SLOT, the corresponding error is generated and output at ERR and ERNO.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs. If the block detects an Ethernet coupler at the specified SLOT, this is indicated by DONE = TRUE and ERR = FALSE and the read IP address is output.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected, the IP address of which shall be read and reported.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

IP_ADR DWORD (IP address)

Output IP_ADR displays the own IP address of the coupler. Each byte in IP_ADR represents one octet of the address.

Example:

IP address 192.15.24.2,
IP_ADR (hex) 16#C00F1802,
IP_ADR (dec) 3222214658

Function call in IL

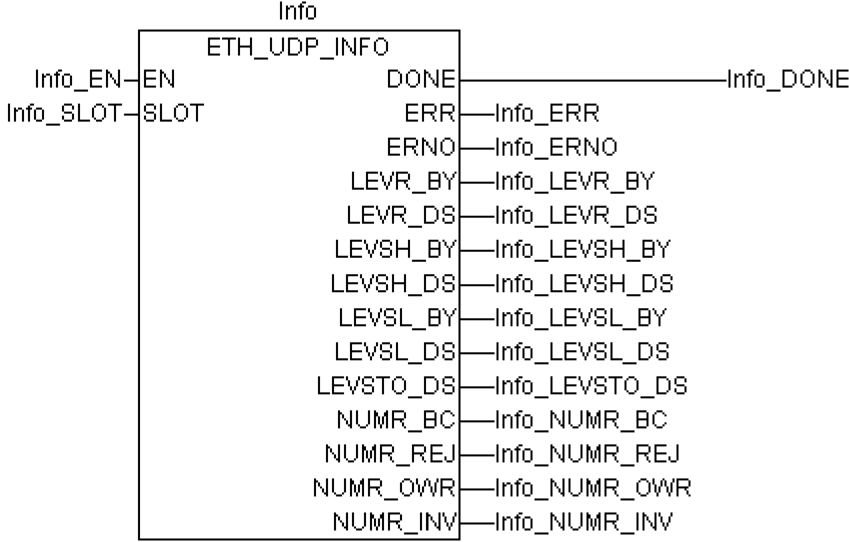
```
CAL  OwnIp(EN := OwnIp_EN,
     SLOT := OwnIp_SLOT)

LD   OwnIp.DONE
ST   OwnIp_DONE

LD   OwnIp.ERR
ST   OwnIp_ERR

LD   OwnIp.ERNO
ST   OwnIp_ERNO

LD   OwnIp.IP_ADR
ST   OwnIp_IP_ADR
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
OwnIp(EN := OwnIp_EN,
      SLOT := OwnIp_SLOT);

OwnIp_DONE := OwnIp.DONE;
OwnIp_ERR := OwnIp.ERR;
OwnIp_ERNO := OwnIp.ERNO;
OwnIp_IP_ADR := OwnIp.IP_ADR;
```

ETH_UDP_INFO Reading status information from the UDP/IP processing



The block ETH_UDP_INFO reads the status information of the UDP/IP processing.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Ethernet_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		ETH_UDP_INFO	Instance name
EN	Input	BOOL	Initialization of the Ethernet UDP/IP data exchange with a FALSE/TRUE edge. A TRUE/FALSE edge stops the processing.
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Initialization completed or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
LEVR_BY	Output	WORD	Filling level of the receive buffer in bytes
LEVR_DS	Output	WORD	Filling level of the receive buffer in data sets
LEVSH_BY	Output	WORD	Filling level of the send buffer for high priority in bytes
LEVSH_DS	Output	WORD	Filling level of the send buffer for high priority in data sets
LEVSL_BY	Output	WORD	Filling level of the send buffer for low priority in bytes
LEVSL_DS	Output	WORD	Filling level of the send buffer for low priority in data sets

LEVSTO_DS	Output	WORD	Filling level of the timeout data packages buffer in data sets
NUMR_BC	Output	DWORD	Number of the received broadcast telegrams
NUMR_REJ	Output	DWORD	Number of data sets discarded during reception
NUMR_OWR	Output	DWORD	Number of data sets overwritten during reception
NUMR_INV	Output	DWORD	Number of received faulty telegrams

Description

Using the ETH_UDP_INFO block, various status information about the UDP/IP processing can be read.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs. If the block detects an Ethernet coupler with activated UDP/IP functionality in the specified SLOT, this is indicated by DONE = TRUE and ERR = FALSE. The corresponding status information are then available at the block outputs.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by this block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

Output ERR is set to TRUE for one cycle.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

LEVR_BY WORD (level of the receive buffer in bytes)

As long as EN = TRUE, output LEVR_BY displays the filling level of the receive buffer in bytes.

LEVR_DS WORD (level of the receive buffer in data sets)

As long as EN = TRUE, output LEVR_DS displays the filling level of the receive buffer in data sets.

LEVSH_BY WORD (level of the send buffer - high priority in bytes)

As long as EN = TRUE, output LEVSH_BY displays the filling level of the high priority send buffer in bytes.

LEVSH_DS WORD (level of the send buffer - high priority in data sets)

As long as EN = TRUE, output LEVSH_DS displays the filling level of the high priority send buffer in data sets.

LEVSL_BY WORD (level of the send buffer - low priority in bytes)

As long as EN = TRUE, output LEVSL_BY displays the filling level of the low priority send buffer in bytes.

LEVSL_DS WORD (level of the send buffer - low priority in data sets)

As long as EN = TRUE, output LEVSL_DS displays the filling level of the low priority send buffer in data sets.

LEVSTO_DS WORD (level of the send buffer - timeout in data sets)

As long as EN = TRUE, output LEVSTO_DS displays the filling level of the timeout buffer in data sets.

NUMR_BC DWORD (number of received broadcasts)

NUMR_BC outputs the number of broadcasts (data packages to all stations) which were received by this station.

NUMR_REJ DWORD (number of receipts rejected)

At output NUMR_REJ, the number of data sets is displayed which were discarded during reception due to a full receive buffer. Data sets are only discarded if this is set accordingly within the configuration of the UDP/IP processing (see Configuration of UDP/IP processing).

NUMR_OWR DWORD (number of receipts overwritten)

At output NUMR_OWR, the number of data sets is displayed which were overwritten during reception due to a full receive buffer. Data sets are only overwritten in the receive buffer, if this is set accordingly within the configuration of the UDP/IP processing (see Configuration of UDP/IP processing).

NUMR_INV DWORD (number of receipts invalid)

NUMR_INV outputs the number of telegrams which were received faulty by this station.

Function call in IL

```
CAL  Info(EN := Info_EN,
     SLOT := Info_SLOT;

LD   Info.DONE
ST   Info_DONE

LD   Info.ERR
ST   Info_ERR

LD   Info.ERNO
ST   Info_ERNO

LD   Info.LEVR_BY
ST   Info_LEVR_BY

LD   Info.LEVR_DS
ST   Info_LEVR_DS
```

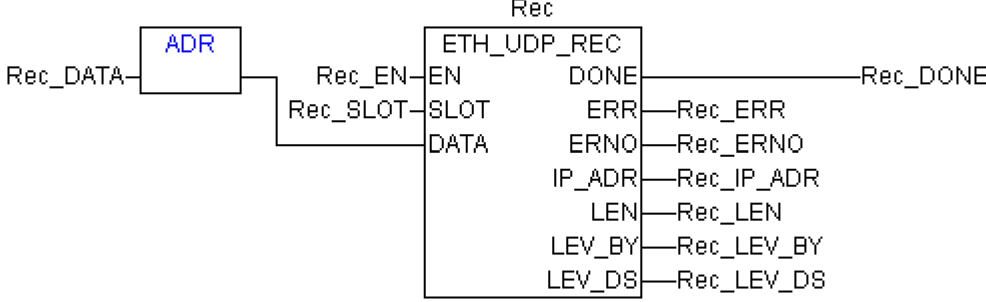
```
LD   Info.LEVSH_BY
ST   Info_LEVSH_BY

LD   Info.LEVSH_DS
ST   Info_LEVSH_DS

LD   Info.LEVSL_BY
ST   Info_LEVSL_BY

LD   Info.LEVSL_DS
ST   Info_LEVSL_DS

LD   Info.LEVSTO_DS
ST   Info_LEVSTO_DS

LD   Info.NUMR_BC
ST   Info_NUMR_BC

LD   Info.NUMR_REJ
ST   Info_NUMR_REJ

LD   Info.NUMR_OWR
ST   Info_NUMR_OWR

LD   Info.NUMR_INV
ST   Info_NUMR_INV
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
Info  (EN := Info_EN,
       SLOT := Info_SLOT;

Info_DONE := Info.DONE;
Info_ERR := Info.ERR;
Info_ERNO := Info.ERNO;
Info_LEVR_BY := Info.LEVR_BY;
Info_LEVR_DS := Info.LEVR_DS;
Info_LEVSH_BY := Info.LEVSH_BY;
Info_LEVSH_DS := Info.LEVSH_DS;
Info_LEVSL_BY := Info.LEVSL_BY;
Info_LEVSL_DS := Info.LEVSL_DS;

Info_LEVSTO_DS := Info.LEVSTO_DS;

Info_NUMR_BC := Info.NUMR_BC;
Info_NUMR_REJ := Info.NUMR_REJ;
Info_NUMR_OWR := Info.NUMR_OWR;
Info_NUMR_INV := Info.NUMR_INV;
```

ETH_UDP_REC Reading a data package from the UDP/IP receive buffer



The ETH_UDP_REC block reads the next data record from the UDP/IP receive buffer and stores the user data to the configured memory area.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Ethernet_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		ETH_UDP_REC	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DATA	Input	DWORD	Variable in which the received user data shall be stored. The variable has to be of the type ARRAY or STRUCT.
DONE	Output	BOOL	Data package stored or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
IP_ADR	Output	DWORD	IP address of the sending device
LEN	Output	WORD	Data package length in bytes
LEV_BY	Output	WORD	Receive buffer filling level in bytes
LEV_DS	Output	WORD	Receive buffer filling level in data records

Description

The operating system reads the received UDP/IP data packages from the ETHERNET coupler and stores them in the receive buffer. The buffer size is determined using the PLC configuration. The data packages are stored with variable lengths. For example, a data package consisting of 16 bytes of user data occupies exactly 22 bytes in the receive buffer (4 bytes for the IP address of the sending device, 2 bytes for the packet length and 16 bytes of user data).

Using the ETH_UDP_REC block, exactly one data package is read. The user data are stored in the configured memory area (DATA). The address of the sending device and the data package length are supplied at the outputs IP_ADR and LEN. DONE = TRUE and ERR = FALSE indicate that the reading process was successful. If an error was detected during block processing, the error is additionally indicated at the outputs ERR and ERNO. Furthermore, the block provides information about the receive buffer filling level displayed in bytes (LEVR_BY) and data records (LEVR_DS).

EN BOOL (enable)

Reading of the receive buffer is performed depending on the signal applied at input EN.

The following applies:
EN = FALSE: Do not read receive buffer
EN = TRUE: Read receive buffer

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by this block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DATA DWORD (data)

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

CAUTION: Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

DONE BOOL (done)

Output DONE indicates that the user data of a data package were copied consecutively from the receive buffer to the memory area of the variable specified at DATA or that the block processing was aborted due to an occurred error. This is why the output has always to be considered together with output ERR.

Output DONE is set to TRUE for one cycle.

The following applies:

DONE = FALSE
ERR = xxx:
The data package was not read from the receive buffer.

DONE = TRUE
ERR = FALSE:
The data package was successfully read from the receive buffer.

DONE = TRUE
ERR = TRUE:
An error occurred while reading the user data from the receive buffer. The user data were not copied to the area specified at DATA . The error can be evaluated at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

Output ERR is set to TRUE for one cycle.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

IP_ADR DWORD (IP address)

Output IP_ADR displays the IP address of the sending device which transmitted the received data package. Each byte in IP_ADR represents one octet of the address.

Example:
IP address 192.15.24.2,
IP_ADR (hex) 16#C00F1802,
IP_ADR (dec) 3222214658

LEN WORD (length)

Output LEN displays the length of the received data package in bytes.

LEV_BY WORD (level in bytes)

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data package occupies output LEN + 6 bytes in the receive buffer (4 bytes for the IP address of the sending device, 2 bytes for the specification of the length).

LEV_DS WORD (level in data sets)

Output LEV_DS displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in IL

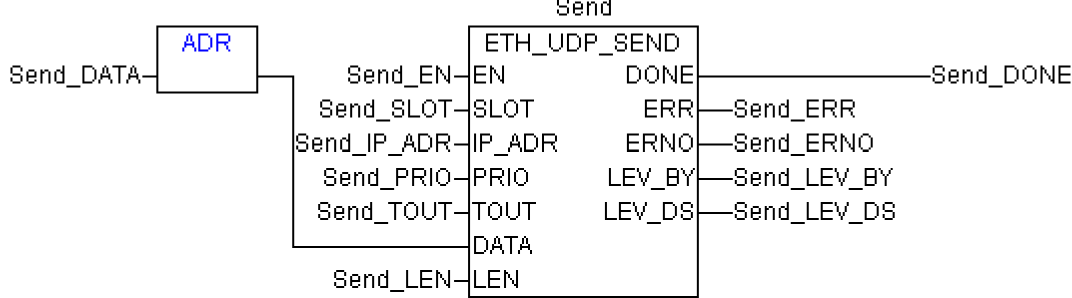
```
LD   Rec_DATA
ADR
ST   Rec.DATA

CAL  Rec(EN := Rec_EN,
     SLOT := Rec_SLOT)

LD   Rec.DONE
ST   Rec_DONE

LD   Rec.ERR
ST   Rec_ERR

LD   Rec.ERNO
ST   Rec_ERNO

LD   Rec.IP_ADR
ST   Rec_IP_ADR

LD   Rec.LEN
ST   Rec_LEN

LD   Rec.LEV_BY
ST   Rec_LEV_BY

LD   Rec.LEV_DS
ST   Rec_LEV_DS
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
REC  (EN := Rec_EN,
     SLOT := Rec_SLOT,
     DATA := ADR(Rec_DATA));

REC_DONE := Rec.DONE;
REC_ERR := Rec.ERR;
REC_ERNO := Rec.ERNO;
REC_IP_ADR := Rec.IP_ADR;
REC_LEN := Rec.LEN;
REC_LEVR_BY := Rec.LEV_BY;
REC_LEVR_DS := Rec.LEV_DS;
```

ETH_UDP_SEND Sending a data package to a station via Ethernet UDP/IP



The block ETH_UDP_SEND is used to transmit data packages via the UDP/IP protocol of the ETHERNET coupler.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Ethernet_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		ETH_UDP_SEND	Instance name
EN	Input	BOOL	Enabling of data package transmission
SLOT	Input	BYTE	Slot (module number) of the coupler
IP_ADR	Input	DWORD	Target IP address to which the data shall be transmitted
PRIO	Input	BOOL	Transmission priority of the data package FALSE = low TRUE = high
TOUT	Input	WORD	Timeout period of the data package in ms
DATA	Input	DWORD	Address of the variable from which on the data shall be copied into the transmit buffer. The variable has to be of the type ARRAY or STRUCT.
LEN	Input	WORD	Number of user data to be transmitted (in bytes)
DONE	Output	BOOL	Data package stored in transmit buffer or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
LEV_BY	Output	WORD	Filling level (in bytes) of the transmit buffer for low/high priority (depending on input PRIO)
LEV_DS	Output	WORD	Filling level (in data records) of the transmit buffer for low/high priority (depending on input PRIO)

Description

The block ETH_UDP_SEND is used to transmit data packages via the UDP/IP protocol of the ETHERNET coupler. The specified packages are stored in the transmit buffer selected by input PRIO. From there, the operating system hands over the data packages to the ETHERNET coupler in order to transmit them to the target address specified at input IP_ADR. The transmit buffer size is determined using the ETH_UDP_INIT block. Using input TOUT, the timeout period can be specified. If TOUT <> 0, the UDP/IP data exchange is automatically performed with receive acknowledgement. If TOUT = 0, no acknowledgement is expected. Output DONE indicates that the specified data package has been stored in the transmit buffer or that an error occurred during block processing. If an error was detected during block processing, the error is additionally indicated at the outputs ERR and ERNO. In case of an error, the data package has to be transmitted again.

The block ETH_UDP_SEND cannot store data packages to the transmit buffer until the ETHERNET UDP/IP processing is set in the controller configuration (see Controller configuration of UDP/IP processing).

EN BOOL (enable)

If a FALSE > TRUE edge is applied to input EN, the specified package is stored to the transmit buffer and then transmitted.

The following applies:

EN = FALSE: The specified packet is not stored in the transmit buffer and therefore not transmitted.

EN = FALSE/TRUE edge: The specified packet is stored in the transmit buffer and transmitted.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by this block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

IP_ADR DWORD (IP address)

At this input, the IP address of the recipient is specified. Each byte in IP_ADR represents one octet of the address.

Example:
IP address 192.15.24.2,
IP_ADR (hex) 16#C00F1802,
IP_ADR (dec) 3222214658

PRIO BOOL (priority)

Input PRIO is used to specify the transmit priority of the data package.

The following applies:

PRIO = FALSE:
The specified data package has low priority. Thus, it is stored in the low priority transmit buffer. All outputs refer to this buffer.

PRIO = TRUE:
The specified data package has high priority. Thus, it is stored in the high priority transmit buffer. All outputs refer to this buffer.

TOUT WORD (timeout)

Using input TOUT, the timeout period can be specified. If TOUT <> 0, the UDP/IP data exchange is automatically performed with receive acknowledgement. If a data package cannot be transmitted within this period (no acknowledge telegram is received), transmission is aborted and the package is lost.

In this case, some distinctive bytes of the data package (see Configuration of UDP/IP processing in CoDeSys) are stored to the timeout buffer and can then be read using the block ETH_UDP_STO.

If TOUT = 0, no acknowledgement is expected.

The following applies:

TOUT = 0:
Data exchange without receive acknowledgement. No data are written to the timeout buffer.

TOUT <> 0:
Data exchange with receive acknowledgement. Each transmitted data record is acknowledged by the recipient. If no acknowledge telegram is received within the set timeout period (in ms), the data are written to the timeout buffer.

DATA DWORD (data)

At input DATA, the address of the variable is specified the data of which are transmitted as user data in this package. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

LEN WORD (length)

At input LEN, the number of user data bytes is specified for the specified package.

The following applies: 1 ≤ LEN ≤ 1464

DONE BOOL (done)

Output DONE indicates that the specified package has been stored in the transmit buffer or that an error occurred during block processing. This is why the output has always to be considered together with output ERR.

Output DONE is set to TRUE for one cycle.

The following applies:

DONE = FALSE
ERR = xxx:
The specified packet has not been stored in the transmit buffer.

DONE = TRUE
ERR = FALSE:
The specified packet has been stored in the transmit buffer.

DONE = TRUE
ERR = TRUE:
An error occurred during transmission. The specified data packet was not stored in the transmit buffer. The error can be evaluated at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

Output ERR is set to TRUE for one cycle.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

LEV_BY WORD (level in bytes)

Output LEV_BY displays the filling level (in bytes) of the transmit buffer selected at input PRIO. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data package occupies output LEN + 8 bytes in the transmit buffer (4 bytes for the IP address of the recipient, 2 bytes for the specification of the length and 2 bytes for the timeout period).

LEV_DS WORD (level in data sets)

Output LEV_DS displays the filling level (in data records) of the transmit buffer selected at input PRIO. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in IL

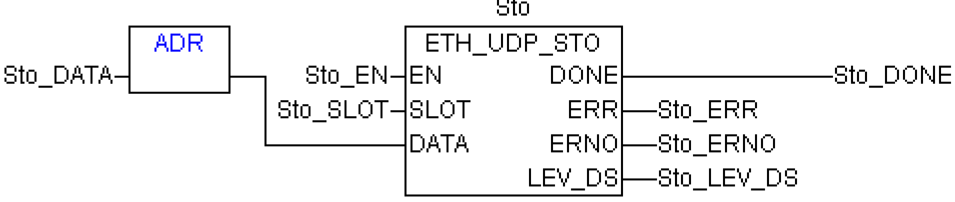
```
LD   Send_DATA
ADR
ST   Send.DATA

CAL  Send(EN := Send_EN,
     SLOT := Send_SLOT,
     IP_ADR := Send_IP_ADR,
     PRIO := Send_PRIO,
     TOUT := Send_TOUT,
     LEN := Send_LEN)

LD   Send.DONE
ST   Send_DONE

LD   Send.ERR
ST   Send_ERR

LD   Send.ERNO
ST   Send_ERNO

LD   Send.LEV_BY
ST   Send_LEV_BY

LD   Send.LEV_DS
ST   Send_LEV_DS
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
Send (EN := Send_EN,
     SLOT := Send_SLOT,
     IP_ADR := Send_IP_ADR,
     PRIO := Send_PRIO,
     TOUT := Send_TOUT,
     DATA := ADR(Send_DATA),
     LEN := Send_LEN);

Send_DONE := Send.DONE;
Send_ERR := Send.ERR;
Send_ERNO := Send.ERNO;
Send_LEV_BY := Send.LEV_BY;
Send_LEV_DS := Send.LEV_DS;
```

ETH_UDP_STO Reading Ethernet UDP/IP timeout data packages from the timeout data buffer



The ETH_UDP_STO block reads lost data packages from the timeout data buffer and stores the user data to the specified memory area.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Ethernet_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		ETH_UDP_STO	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DATA	Input	DWORD	Variable in which the data of the timeout package are stored. The variable has to be of the type ARRAY or STRUCT.
DONE	Output	BOOL	Data package stored or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
LEV_DS	Output	WORD	Timeout buffer filling level in data records

Description

During the transmission of a data package, the success of the transmission is monitored by an adjustable timeout period. When this time is exceeded, distinctive information of the data package are stored in the timeout buffer.

These are:

- the IP address of the receiver (4 bytes)
- header data of the data set (the number is specified with the controller configuration of the UDP/IP processing).

The buffer length can as well be set using the controller configuration of the UDP/IP processing. The buffer is constructed as a circular buffer (FIFO). If the buffer is full, the oldest entry in the buffer is overwritten. When a rising edge occurs at input EN, the ETH_UDP_STO block verifies whether a data

package is stored in the buffer and makes the information mentioned above available for the user (starting at the variable specified at input DATA).

The block ETH_UDP_STO cannot be used until the ETHERNET UDP/IP processing is set in the controller configuration (see Controller configuration of UDP/IP processing). Additionally, input TOUT of the transmit block ETH_UDP_SEND must be <> 0.

EN BOOL (enable)

Reading of the timeout buffer is performed depending on the signal state applied at input EN.

The following applies:

EN = FALSE:
The timeout buffer is not read.

EN = TRUE:
The timeout buffer is read.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by this block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DATA DWORD (data)

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

CAUTION: Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

DONE BOOL (done)

Output DONE indicates that the information of a data package was copied consecutively from the timeout buffer to the memory area of the variable specified at DATA or that the block processing was aborted due to an occurred error. This is why the output has always to be considered together with output ERR.

Output DONE is set to TRUE for one cycle.

The following applies:

DONE = FALSE
ERR = xxx:
The information of a data package in the timeout buffer have not been read.

DONE = TRUE
ERR = FALSE:
The information of a data package in the timeout buffer have been read.

DONE = TRUE
ERR = TRUE:
An error occurred while reading information from a data package stored in the timeout buffer. The user data were not copied to the area specified at DATA. The error can be evaluated at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

Output ERR is set to TRUE for one cycle.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

LEV_DS WORD (level in data sets)

Output LEV_DS displays the filling level (in data records) of the timeout buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in IL

```
LD   Sto_DATA
ADR
ST   Sto.DATA

CAL  Sto(EN := Sto_EN, SLOT := Sto_SLOT)

LD   Sto.DONE
ST   Sto_DONE

LD   Sto.ERR
ST   Sto_ERR

LD   Sto.ERNO
ST   Sto_ERNO

LD   Sto.LEV_DS
ST   Sto_LEV_DS
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
Sto  (EN := Sto_EN,
     SLOT := Sto_SLOT,
     DATA := ADR(Sto_DATA));

Sto_DONE := Sto.DONE;
Sto_ERR := Sto.ERR;
Sto_ERNO := Sto.ERNO;
Sto_LEV_DS := Sto.LEV_DS;
```

IP_ADR_DWORD_TO_STRING Format conversion of the IP address

```
                       IP_ADR_DWORD_TO_STRING
IP_ADR_DWORD—IP_ADR                                    ——IP_ADR_STRING
```

The block IP_ADR_DWORD_TO_STRING converts an IP address given in the DWORD format into an IP address in the STRING format.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Ethernet_AC500_V10.LIB	

Block type

Function

Parameters

IP_ADR	Input	DWORD	IP address in DWORD format
	Output	STRING(16)	Converted IP address

Description

The IP address of an Ethernet device can be represented in different ways. One of the best known formats of an IP address is the STRING format, composed of four numbers between 0 and 255, separated by dots (e.g. 192.15.24.2). The block IP_ADR_DWORD_TO_STRING converts an IP address given in the DWORD format into an IP address in the STRING format.

IP_ADR DWORD (IP address)

At input IP_ADR, the IP address in DWORD format is specified.

(Output) STRING(16)

The output of the block IP_ADR_DWORD_TO_STRING outputs the converted IP address in STRING(16) format.

Example:

The following value is applied at block input IP_ADR:

IP_ADR_WORD: (hex) 16#C00F1802,
or
IP_ADR_WORD: (dec) 3222214658

The converted value is displayed at the block output:

IP_ADR_STRING: '192.15.24.2'

Function call in IL

```
LD   IP_ADR_DWORD
IP_ADR_DWORD_TO_STRING
ST   IP_ADR_STRING
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
IP_ADR_STRING := IP_ADR_DWORD_TO_STRING(IP_ADR_DWORD);
```

IP_ADR_STRING_TO_DWORD Format conversion of the IP address

```
                              IP_ADR_STRING_TO_DWORD
IP_ADR_STRING—IP_ADR                                            ——IP_ADR_DWORD
```

The block IP_ADR_STRING_TO_DWORD converts an IP address given in the STRING format into an IP address in the DWORD format.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Ethernet_AC500_V10.LIB	

Block type

Function

Parameters

IP_ADR	Input	STRING(16)	IP address in STRING format
	Output	DWORD	Converted IP address

Description

The IP address of an Ethernet device can be represented in different ways. One of the best known formats of an IP address is the STRING format, composed of four numbers between 0 and 255, separated by dots (e.g. 192.15.24.2). The block IP_ADR_STRING_TO_DWORD converts an IP address given in the STRING format into an IP address in the DWORD format.

IP_ADR STRING(16) (IP address)

At input IP_ADR, the IP address in STRING(16) format is specified.

(Output) DWORD

The output of the block IP_ADR_STRING_TO_DWORD outputs the converted IP address in DWORD format.

Example:

The following value is applied at block input IP_ADR:

IP_ADR_STRING: '192.15.24.2'

The converted value is displayed at the block output:

IP_ADR_WORD: (hex) 16#C00F1802
or
IP_ADR_WORD: (dec) 3222214658

Function call in IL

```
LD   IP_ADR_STRING
IP_ADR_STRING_TO_DWORD
ST   IP_ADR_DWORD
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
IP_ADR_DWORD := IP_ADR_STRING_TO_DWORD(IP_ADR_STRING);
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

Components of the Ethernet library 3

E

ETH_MOD_INFO Reading status information from the OpenModbus on TCP/IP processing 5

ETH_MOD_MAST Processing OpenModbus on TCP/IP Client (master) telegrams 8

ETH_OWN_IP Outputting the own IP address 12

ETH_UDP_INFO Reading status information from the UDP/IP processing 15

ETH_UDP_REC Reading a data package from the UDP/IP receive buffer 19

ETH_UDP_SEND Sending a data package to a station via Ethernet UDP/IP 23

ETH_UDP_STO Reading Ethernet UDP/IP timeout data packages from the timeout data buffer 28

IP_ADR_DWORD_TO_STRING Format conversion of the IP address 31

IP_ADR_STRING_TO_DWORD Format conversion of the IP address 33

G

Glossary 35

O

Overview of blocks arranged according to their call names 4

P

Preconditions for the use of the library 2

S

Special characteristics of the Ethernet library 2

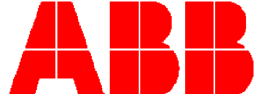
Software Description

AC500

Scalable PLC
for Individual Automation

ARCNET
Function Block Library

ARCNET



Contents

ARCNET Library 2

Preconditions for the use of the library 2

Components of the ARCNET library 2

Overview of blocks arranged according to their call names 2

ARC_INFO Reading status information from the ARCNET processing 3

ARC_OWN_NODE Outputting the own NODE ID 7

ARC_REC Reading a data package from the ARCNET receive buffer 9

ARC_SEND Sending a data package to a station via ARCNET 13

ARC_STO Reading ARCNET timeout data packages from the timeout data buffer 17

Glossary 21

Index 23

ARCNET Library

Preconditions for the use of the library

Note:

The blocks of the ARCNET Library can only be executed in RUN mode of the PLC, but not in simulation mode.

Before an AC500 coupler can be set into operation, it must be configured with the PLC Configuration within the Control Builder. The use of the ARCNET library blocks (ARC_INFO, ARC_REC, ARC_SEND; ARC_STO) presupposes that the ARCNET coupler is configured in the ARCNET data exchange mode.

Components of the ARCNET Library

The library "ARCNET_AC500_V12.lib" contains the following function blocks:

Group: Data	
ARC_INFO	Reading status information from the ARCNET processing
ARC_REC	Reading a data package from the ARCNET receive buffer
ARC_SEND	Sending a data package to a station via ARCNET
ARC_STO	Reading ARCNET timeout data packagesfrom the timeout data buffer

Group: General	
ARC_OWN_NODE	Outputting the own NODE ID

Overview of blocks, arranged according to their call names

Used abbreviations:

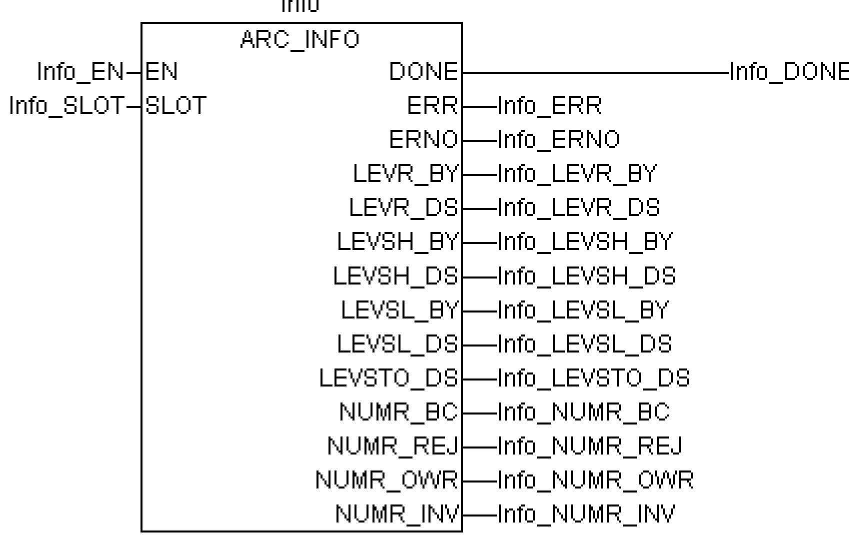
FBhv ... Function block with historical values

FBnohv ... Function block without historical values

F … Function

CE name	Type	Function
ARC_INFO	FBhv	Reading status information from the ARCNET processing
ARC_OWN_NODE	FBnohv	Outputting the own NODE ID
ARC_REC	FBhv	Reading a data package from the ARCNET receive buffer
ARC_SEND	FBhv	Sending a data package to a station via ARCNET
ARC_STO	FBhv	Reading ARCNET timeout data packagesfrom the timeout data buffer

ARC_INFO Reading status information from the ARCNET processing



The block ARC_INFO reads the status information of the ARCNET processing.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	ARCNET_AC500_V12.LIB	

Block type

Function block with historical values

Parameters

Instance		ARC_INFO	Instance name
EN	Input	BOOL	Enable of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
LEVR_BY	Output	WORD	Filling level of the receive buffer in bytes
LEVR_DS	Output	WORD	Filling level of the receive buffer in data sets
LEVSH_BY	Output	WORD	Filling level of the send buffer for high priority in bytes
LEVSH_DS	Output	WORD	Filling level of the send buffer for high priority in data sets
LEVSL_BY	Output	WORD	Filling level of the send buffer for low priority in bytes
LEVSL_DS	Output	WORD	Filling level of the send buffer for low priority in data sets
LEVSTO_DS	Output	WORD	Filling level of the timeout data packages buffer in data sets
NUMR_BC	Output	DWORD	Number of the received broadcast telegrams
NUMR_REJ	Output	DWORD	Number of data sets discarded during reception

NUMR_OWR	Output	DWORD	Number of data sets overwritten during reception
NUMR_INV	Output	DWORD	Number of received faulty telegrams

Description

Using the ARC_INFO block, various status information about the ARCNET processing can be read. This information can only be read if in the PLC Configuration of the Control Builder "ARCNET data exchange" was selected.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs. If the block detects a coupler with an activated ARCNET functionality in the specified SLOT, this is indicated by DONE = TRUE and ERR = FALSE. The corresponding status information are then available at the block outputs.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by this block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

LEVR_BY WORD (level of the receive buffer in bytes)

As long as EN = TRUE, output LEVR_BY displays the filling level of the receive buffer in bytes.

LEVR_DS WORD (level of the receive buffer in data sets)

As long as EN = TRUE, output LEVR_DS displays the filling level of the receive buffer in data sets.

LEVSH_BY WORD (level of the send buffer - high priority in bytes)

As long as EN = TRUE, output LEVSH_BY displays the filling level of the high priority send buffer in bytes.

LEVSH_DS WORD (level of the send buffer - high priority in data sets)

As long as EN = TRUE, output LEVSH_DS displays the filling level of the high priority send buffer in data sets.

LEVSL_BY WORD (level of the send buffer - low priority in bytes)

As long as EN = TRUE, output LEVSL_BY displays the filling level of the low priority send buffer in bytes.

LEVSL_DS WORD (level of the send buffer - low priority in data sets)

As long as EN = TRUE, output LEVSL_DS displays the filling level of the low priority send buffer in data sets.

LEVSTO_DS WORD (level of the send buffer - timeout in data sets)

As long as EN = TRUE, output LEVSTO_DS displays the filling level of the timeout buffer in data sets.

NUMR_BC DWORD (number of received broadcasts)

NUMR_BC outputs the number of broadcasts (data packages to all stations) which were received by this station.

NUMR_REJ DWORD (number of receipts rejected)

At output NUMR_REJ, the number of data sets is displayed which were discarded during reception due to a full receive buffer. Data sets are only discarded if this is set accordingly within the configuration of the ARCNET processing (see System Technology, chapter 3.6.1.2 The internal ARCNET coupler PM5x1-ARCNET).

NUMR_OWR DWORD (number of receipts overwritten)

At output NUMR_OWR, the number of data sets is displayed which were overwritten during reception due to a full receive buffer. Data sets are only overwritten in the receive buffer, if this is set accordingly within the configuration of the ARCNET processing (see System Technology, chapter 3.6.1.2 The internal ARCNET coupler PM5x1-ARCNET).

NUMR_INV DWORD (number of receipts invalid)

NUMR_INV outputs the number of telegrams which were received faulty by this station.

Function call in IL

```
CAL  Info(EN := Info_EN,
     SLOT := Info_SLOT;

LD   Info.DONE
ST   Info_DONE

LD   Info.ERR
ST   Info_ERR

LD   Info.ERNO
ST   Info_ERNO

LD   Info.LEVR_BY
ST   Info_LEVR_BY

LD   Info.LEVR_DS
ST   Info_LEVR_DS

LD   Info.LEVSH_BY
ST   Info_LEVSH_BY

LD   Info.LEVSH_DS
ST   Info_LEVSH_DS

LD   Info.LEVSL_BY
ST   Info_LEVSL_BY
```

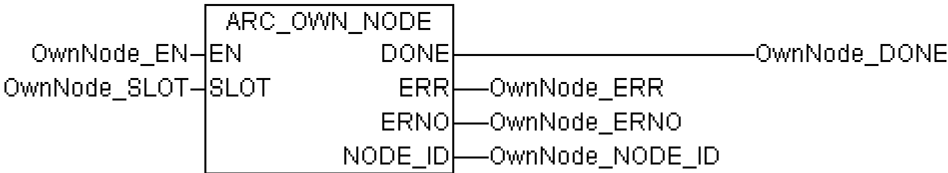
```
LD   Info.LEVSL_DS
ST   Info_LEVSL_DS

LD   Info.LEVSTO_DS
ST   Info_LEVSTO_DS

LD   Info.NUMR_BC
ST   Info_NUMR_BC

LD   Info.NUMR_REJ
ST   Info_NUMR_REJ

LD   Info.NUMR_OWR
ST   Info_NUMR_OWR

LD   Info.NUMR_INV
ST   Info_NUMR_INV
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
Info  (EN := Info_EN,
       SLOT := Info_SLOT;

Info_DONE := Info.DONE;
Info_ERR := Info.ERR;
Info_ERNO := Info.ERNO;
Info_LEVR_BY := Info.LEVR_BY;
Info_LEVR_DS := Info.LEVR_DS;
Info_LEVSH_BY := Info.LEVSH_BY;
Info_LEVSH_DS := Info.LEVSH_DS;
Info_LEVSL_BY := Info.LEVSL_BY;
Info_LEVSL_DS := Info.LEVSL_DS;
Info_LEVSTO_DS := Info.LEVSTO_DS;

Info_NUMR_BC := Info.NUMR_BC;
Info_NUMR_REJ := Info.NUMR_REJ;
Info_NUMR_OWR := Info.NUMR_OWR;
Info_NUMR_INV := Info.NUMR_INV;
```

ARC_OWN_NODE Outputting the own NODE ID



The block ARC_OWN_NODE outputs the NODE ID address of the ARCNET coupler installed at slot SLOT.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	ARCNET_AC500_V12.LIB	

Block type

Function block without historical values

Parameters

EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NODE_ID	Output	BYTE	Own NODE ID of the coupler

Description

Prior to setting into operation an AC500 coupler, it must be configured using the system configuration of the Control Builder. One of the parameters of an ARCNET coupler is its own NODE_ID. Using the block ARC_NODE_ID, the latest configured NODE ID of the device at SLOT can be read. If no ARCNET coupler is installed at SLOT, the corresponding error is generated and output at ERR and ERNO.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs. If the block detects an ARCNET coupler at the specified SLOT, this is indicated by DONE = TRUE and ERR = FALSE and the relevant NODE ID is output.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected, from which the NODE ID is to be output.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

NODE_ID BYTE (NODE-ID)

Output NODE_ID displays the own NODE ID of the coupler.

Function call in IL

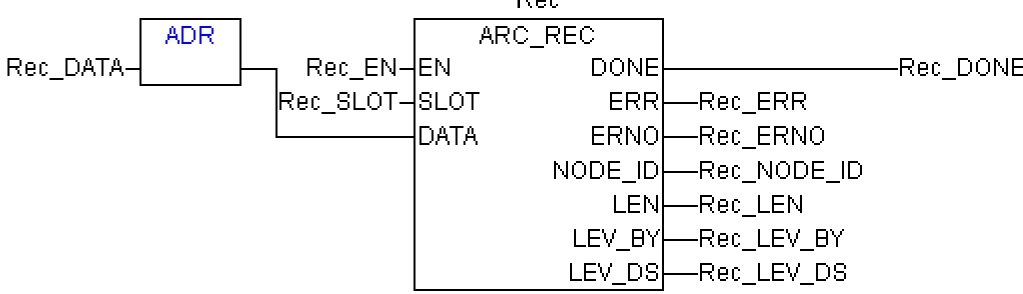
```
CAL  ARC_OWN_NODE(

     EN := OwnNode_EN,
     SLOT := OwnNode_SLOT)

LD   OwnNode.DONE
ST   OwnNode_DONE

LD   OwnNode.ERR
ST   OwnNode_ERR

LD   OwnNode.ERNO
ST   OwnNode_ERNO

LD   OwnNode.NODE_ID
ST   OwnNode_NODE_ID
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
ARC_OWN_NODE(EN := OwnNode_EN, SLOT := OwnNode_SLOT);

OwnNode_DONE := OwnNode.DONE;
OwnNode_ERR := OwnNode.ERR;
OwnNode_ERNO := OwnNode.ERNO;
OwnNode_IP_ADR := OwnNode.NODE_ID;
```

ARC_REC Reading a data package from the ARCNET receive buffer



The ARC_REC block reads the next data record from the ARCNET data exchange receive buffer and stores the user data to the configured memory area.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	ARCNET_AC500_V12.LIB	

Block type

Function block with historical values

Parameters

Instance		ARC_REC	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DATA	Input	DWORD	Variable in which the received user data are to be stored. The variable has to be of the type ARRAY or STRUCT.
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NODE_ID	Output	BYTE	NODE ID of the sending device
LEN	Output	WORD	Data package length in bytes
LEV_BY	Output	WORD	Receive buffer filling level in bytes
LEV_DS	Output	WORD	Receive buffer filling level in data records

Description

The operating system reads the received ARCNET data packages from the ARCNET coupler and stores them in the receive buffer. The buffer size is determined using the PLC Configuration of the Control Builder. The data packages are stored with variable lengths. For example, a data package consisting of 16 bytes of user data occupies exactly 22 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the packet length and 16 bytes of user data).

Using the ARC_REC block, exactly one data package is read. The user data are stored in the configured memory area (DATA). The NODE ID of the sending device and the data package length are supplied at the outputs NODE_ID and LEN. DONE = TRUE and ERR = FALSE indicate that the reading process

was successful. If an error was detected during block processing, the error is additionally indicated at the outputs ERR and ERNO. Furthermore, the block provides information about the receive buffer filling level displayed in bytes (LEVR_BY) and data records (LEVR_DS).

Before the ARC_REC block can read data packages from the receive buffer, the setting "ARCNET data exchange" must have been selected in the PLC Configuration of the Control Builder.

EN BOOL (enable)

Reading of the receive buffer is performed depending on the signal applied at input EN.

The following applies:
EN = FALSE: Do not read receive buffer
EN = TRUE: Read receive buffer

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by this block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DATA DWORD (data)

Input DATA is used to specify the address of the variable to which the user data shall be copied. DATA must be the address of a variable of the type ARRAY or STRUCT.

CAUTION: Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

DONE BOOL (done)

Output DONE indicates that the user data of a data package were copied consecutively from the receive buffer to the memory area of the variable specified at DATA or that the block processing was aborted due to an occurred error. This is why the output has always to be considered together with output ERR.

Output DONE is set to TRUE for one cycle.

The following applies:

DONE = FALSE
ERR = xxx:
The data package was not read from the receive buffer.

DONE = TRUE
ERR = FALSE:
The data package was read from the receive buffer.

DONE = TRUE
ERR = TRUE:
An error occurred while reading the user data from the receive buffer. The user data were not copied to the area specified at DATA . The error can be evaluated at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

Output ERR is set to TRUE for one cycle.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

NODE_ID BYTE (NODE-ID)

Output NODE_ID displays the NODE ID of the sending device which transmitted the received data package.

LEN WORD (length)

Output LEN displays the length of the received data package in bytes.

LEV_BY WORD (level in bytes)

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data package occupies output LEN + 6 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the specification of the length).

LEV_DS WORD (level in data sets)

Output LEV_DS displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in IL

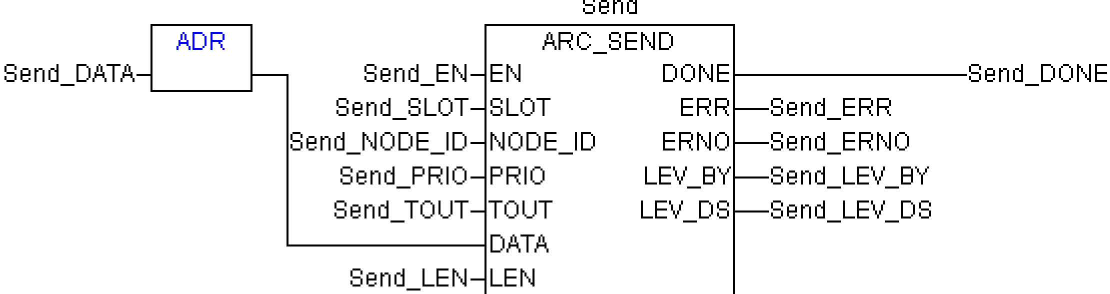
```
LD   Rec_DATA
ADR
ST   Rec.DATA

CAL  Rec(EN := Rec_EN,
     SLOT := Rec_SLOT)

LD   Rec.DONE
ST   Rec_DONE

LD   Rec.ERR
ST   Rec_ERR

LD   Rec.ERNO
ST   Rec_ERNO

LD   Rec.NODE_ID
ST   Rec_NODE_ID

LD   Rec.LEN
ST   Rec_LEN

LD   Rec.LEV_BY
ST   Rec_LEV_BY

LD   Rec.LEV_DS
ST   Rec_LEV_DS
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
REC  (EN := Rec_EN,
     SLOT := Rec_SLOT,
     DATA := ADR(Rec_DATA));

REC_DONE := Rec.DONE;
REC_ERR := Rec.ERR;
REC_ERNO := Rec.ERNO;
REC_NODE_ID := Rec.NODE_ID;
REC_LEN := Rec.LEN;
REC_LEVR_BY := Rec.LEV_BY;
REC_LEVR_DS := Rec.LEV_DS;
```

ARC_SEND Sending a data package to a station via ARCNET



The block ARC_SEND is used to transmit data packages via the ARCNET coupler.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	ARCNET_AC500_V12.LIB	

Block type

Function block with historical values

Parameters

Instance		ARC_SEND	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
NODE_ID	Input	BYTE	NODE ID to which the data shall be transmitted
PRIO	Input	BOOL	Transmission priority of the data package
TOUT	Input	WORD	Timeout period of the data package in ms
DATA	Input	DWORD	Address of the variable from which on the data shall be copied into the transmit buffer. The variable has to be of the type ARRAY or STRUCT.
LEN	Input	WORD	Number of user data to be transmitted (in bytes)
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
LEV_BY	Output	WORD	Filling level (in bytes) of the transmit buffer for low/high priority (depending on input PRIO)
LEV_DS	Output	WORD	Filling level (in data records) of the transmit buffer for low/high priority (depending on input PRIO)

Description

The block ARC_SEND is used to transmit data packages via the ARCNET coupler. The specified packages are stored in the transmit buffer which is selected by input PRIO. From there, the operating system hands over the data packages to the ARCNET coupler in order to transmit them to the target address specified at input NODE_ID. The transmit buffer size is determined using the PLC Configuration of the Control Builder. Using input TOUT, the timeout period can be specified. If TOUT <> 0, the ARCNET data exchange is automatically performed with receive acknowledgement. If TOUT = 0, no acknowledgement is expected. Output DONE indicates that the specified data package has been stored in the transmit buffer or that an error occurred during block processing. If an error was detected during block processing, the error is additionally indicated at the outputs ERR and ERNO. In case of an error, the data package has to be transmitted again.

Before the ARC_SEND block can store data packages in the receive buffer, the setting "ARCNET data exchange" must have been selected in the PLC Configuration of the Control Builder.

EN BOOL (enable)

If a FALSE > TRUE edge is applied to input EN, the specified package is stored to the transmit buffer and then transmitted.

The following applies:

EN = FALSE: The specified packet is not stored in the transmit buffer and therefore not transmitted.

EN = FALSE/TRUE edge: The specified packet is stored in the transmit buffer and transmitted.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by this block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

NODE_ID BYTE (NODE-ID)

At this input, the NODE ID of the recipient is specified to which data are to be sent.

PRIO BOOL (priority)

Input PRIO is used to specify the transmit priority of the data package.

The following applies:

PRIO = FALSE:
The specified data package has low priority. Thus, it is stored in the low priority transmit buffer. All outputs refer to this buffer.

PRIO = TRUE:
The specified data package has high priority. Thus, it is stored in the high priority transmit buffer. All outputs refer to this buffer.

TOUT WORD (timeout)

Using input TOUT, the timeout period can be specified. If TOUT <> 0, the ARCNET data exchange is automatically performed with receive acknowledgement. If a data package cannot be transmitted within this period (no acknowledge telegram is received), transmission is aborted and the package is lost.

In this case, some distinctive bytes of the data package (see also the PLC configuration of the Control Builder) are stored to the timeout buffer and can then be read using the block ARC_STO.

If TOUT = 0, no acknowledgement is expected.

The following applies:

TOUT = 0:
Data exchange without receive acknowledgement. No data are written to the timeout buffer.

TOUT <> 0:
Data exchange with receive acknowledgement. Each transmitted data record is acknowledged by the recipient. If no acknowledge telegram is received within the set timeout period (in ms), the data are written to the timeout buffer.

DATA DWORD (data)

At input DATA, the address of the variable is specified the data of which are transmitted as user data in this package. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

LEN WORD (length)

At input LEN, the number of user data bytes is given for the specified package. The maximum value at the input depends on the set format of the ARCNET processing (long or short packet). This is specified in the PLC Configuration of the Control Builder.

The following applies:

"long pocket" enabled: 1 < LEN < 253; 257 < LEN < 508

"long pocket" disabled: 1 < LEN < 253

DONE BOOL (done)

Output DONE indicates that the specified package has been stored in the transmit buffer or that an error occurred during block processing. This is why the output has always to be considered together with output ERR.

Output DONE is set to TRUE for one cycle.

The following applies:

DONE = FALSE
ERR = xxx:
The specified packet has not been stored in the transmit buffer.

DONE = TRUE
ERR = FALSE:
The specified packet has been stored in the transmit buffer.

DONE = TRUE
ERR = TRUE:
An error occurred during transmission. The specified data packet was not stored in the transmit buffer. The error can be evaluated at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

Output ERR is set to TRUE for one cycle.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

LEV_BY WORD (level in bytes)

Output LEV_BY displays the filling level (in bytes) of the transmit buffer selected at input PRIO. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data package occupies output LEN + 8 bytes in the transmit buffer (4 bytes for the NODE ID of the recipient, 2 bytes for the specification of the length and 2 bytes for the timeout period).

LEV_DS WORD (level in data sets)

Output LEV_DS displays the filling level (in data records) of the transmit buffer selected at input PRIO. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in IL

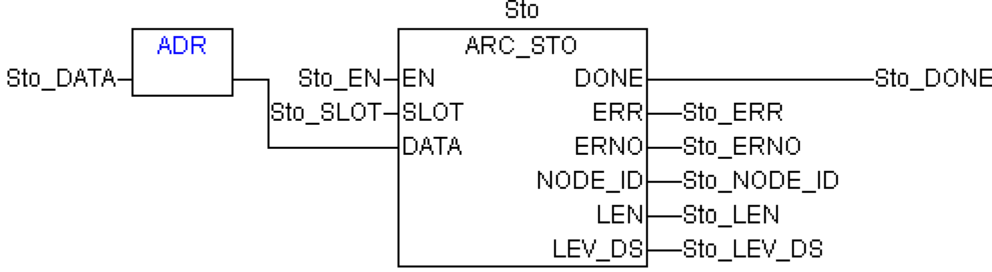
```
LD   Send_DATA
ADR
ST   Send.DATA

CAL  Send(EN := Send_EN,
     SLOT := Send_SLOT,
     IP_ADR := Send_IP_ADR,
     PRIO := Send_PRIO,
     TOUT := Send_TOUT,
     LEN := Send_LEN)

LD   Send.DONE
ST   Send_DONE

LD   Send.ERR
ST   Send_ERR

LD   Send.ERNO
ST   Send_ERNO

LD   Send.LEV_BY
ST   Send_LEV_BY

LD   Send.LEV_DS
ST   Send_LEV_DS
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
Send (EN := Send_EN,
     SLOT := Send_SLOT,
     NODE_ID := Send_NODE_ID,
     PRIO := Send_PRIO,
     TOUT := Send_TOUT,
     DATA := ADR(Send_DATA),
     LEN := Send_LEN);

Send_DONE := Send.DONE;
Send_ERR := Send.ERR;
Send_ERNO := Send.ERNO;
Send_LEV_BY := Send.LEV_BY;
Send_LEV_DS := Send.LEV_DS;
```

ARC_STO Reading ARCNET timeout data packages from the timeout data buffer



The ARC_STO block reads lost data packages from the timeout data buffer and stores the user data to the specified memory area.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Included in library:	ARCNET_AC500_V12.LIB	

Block type

Function block with historical values

Parameters

Instance		ARC_STO	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DATA	Input	DWORD	Variable in which the data of the timeout package are stored. The variable has to be of the type ARRAY or STRUCT.
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NODE_ID	Output	BYTE	Target NODE ID, to which data could not be sent
LEN	Output	WORD	Original length of data, which not could be sent
LEV_DS	Output	WORD	Timeout buffer filling level in data records

Description

During the transmission of a data package, the success of the transmission is monitored by an adjustable timeout period. When this time is exceeded, distinctive information of the data package are stored in the timeout buffer.

These are:

- the NODE ID of the receiver (4 bytes)
- the length of the involved data packet
- header data of the involved data set.

The length of the timeout buffer as well as the number of user data to be stored can be set for the ARCNET processing in the PLC configuration of the Control Builder. The buffer is constructed as a circular buffer (FIFO). If the buffer is full, the oldest entry in the buffer is overwritten. If the ARC_STO block is enabled by EN = TRUE, it checks whether a data packet is stored in the buffer and provides the user with the above mentioned information as of the variable given at input DATA. At the outputs NODE_ID and LEN, the NODE ID and the original length of the telegram, which not could be sent, are available.

Before the ARC_STO block can be used, the setting "ARCNET data exchange" must have been selected in the PLC Configuration of the Control Builder. Additionally, the input TOUT of the ARC_SEND block must be <>0.

EN BOOL (enable)

Reading of the timeout buffer is performed depending on the signal state applied at input EN.

The following applies:

EN = FALSE:
The timeout buffer is not read.

EN = TRUE:
The timeout buffer is read.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected which shall be used by this block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DATA DWORD (data)

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

CAUTION: Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

DONE BOOL (done)

Output DONE indicates that the information of a data package was copied consecutively from the timeout buffer to the memory area of the variable specified at DATA or that the block processing was aborted due to an occurred error. This is why the output has always to be considered together with output ERR.

Output DONE is set to TRUE for one cycle.

The following applies:

DONE = FALSE
ERR = xxx:
The information of a data package in the timeout buffer have not been read.

DONE = TRUE
ERR = FALSE:
The information of a data package in the timeout buffer have been read.

DONE = TRUE
ERR = TRUE:
An error occurred while reading information from a data package stored in the timeout buffer. The user data were not copied to the area specified at DATA. The error can be evaluated at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

Output ERR is set to TRUE for one cycle.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

NODE_ID BYTE (NODE ID)

At the output NODE_ID, the NODE ID to which data could not be sent, is available.

LEN WORD (length)

At the output LEN, the original length of the telegram, which not could be sent, is available.

LEV_DS WORD (level in data sets)

Output LEV_DS displays the filling level (in data records) of the timeout buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in IL

```
LD    Sto_DATA
ADR
ST    Sto.DATA

CAL   Sto(EN := Sto_EN, SLOT := Sto_SLOT)

LD    Sto.DONE
ST    Sto_DONE

LD    Sto.ERR
ST    Sto_ERR

LD    Sto.ERNO
ST    Sto_ERNO

LD    Sto.NODE_ID
ST    Sto_NODE_ID

LD    Sto.LEN
ST    Sto_LEN

LD    Sto.LEV_DS
ST    Sto_LEV_DS
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
Sto   (EN := Sto_EN,
      SLOT := Sto_SLOT,
      DATA := ADR(Sto_DATA));

Sto_DONE := Sto.DONE;
Sto_ERR := Sto.ERR;
Sto_ERNO := Sto.ERNO;
Sto_NODE_ID := Sto.NODE_ID;
Sto_LEN := Sto.LEN;
Sto_LEV_DS := Sto.LEV_DS;
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index ARCNET

A

ARC_INFO Reading status information from the ARCNET processing 3

ARC_OWN_NODE Outputting the own NODE ID 7

ARC_REC Reading a data package from the ARCNET receive buffer 9

ARC_SEND Sending a data package to a station via ARCNET 13

ARC_STO Reading ARCNET timeout data packagesfrom the timeout data buffer 17

C

Components of the ARCNET library 2

G

Glossary 20

O

Overview of blocks arranged according to their call names 2

P

Preconditions for the use of the library 2

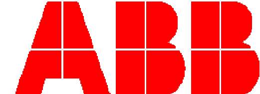
Software Description

AC500

Scalable PLC
for Individual Automation

PROFIBUS DP
Function Block Library

PROFIBUS



Contents

PROFIBUS DP Library 2

Precondition for the use of the PROFIBUS DP Library 2

Components of the PROFIBUS DP Library 2

Overview of blocks arranged according to their call names 4

DPM_CTRL Sending global control commands to the DP slave 5

DPM_READ_INPUT Reading the input data of a slave which is not assigned to the master 11

DPM_READ_OUTPUT Reading the output data of a slave which is not assigned to the master 14

DPM_SET_PRM Sending user parameters to a DP slave 17

DPM_SLV_DIAG Polling detailed diagnostic data of a DP slave 20

DPM_STAT Reading out the status of the PROFIBUS coupler 30

DPM_SYS_DIAG Reading out a status overview of all DP slaves 37

DPV1_MSAC1_READ Reading a data block from a DPV1 slave 40

DPV1_MSAC1_WRITE Writing a data block to a DPV1 slave 45

Glossary 50

Index 52

PROFIBUS DP Library

Preconditions for the use of the PROFIBUS DP Library

The PROFIBUS coupler in the AC500 controller series can solely be operated in operating mode DP master.

The function blocks contained in the PROFIBUS library access to the PLC runtime system as well as directly to the coupler. The required definitions and functions are stored in the external library "SysExt_AC500_V10.lib". This library is automatically included into the corresponding user program during coupler configuration.

Note:

Neither the blocks of the PROFIBUS library nor the PROFIBUS communication can be run in simulation mode. The PROFIBUS communication only runs in the PLC mode RUN, but not in the modes Single Cycle, Step and Breakpoint.

Components of the PROFIBUS DP Library

It is not necessary to include the library for normal cyclic data exchange via PROFIBUS-DP. The library contains additional function blocks which allow an easy handling of the PROFIBUS coupler. Additionally, various data types are defined in this library. These structures enable a clear presentation of data sets.

PROFIBUS DP MASTER

Function blocks

The library "Profibus_AC500_V10.lib" contains the following function blocks:

Group: Diagnosis	
DPM_SLV_DIAG	Polling of detailed diagnostic data of a DP slave
DPM_STAT	Reading out the status of the PROFIBUS coupler
DPM_SYS_DIAG	Reading out the status survey of all DP slaves

Group: Control	
DPM_CTRL	Sending global control commands to the DP slave

Group: Parameter	
DPM_SET_PRM	Sending user parameters to a DP slave

Group: Data	
DPM_READ_INPUT	Reading input data of a slave which is not assigned to the master
DPM_READ_OUTPUT	Reading output data of a slave which is not assigned to the master
Subgroup: DPV1	
DPV1_MSAC1_READ	Reading a data block from a DPV1 slave
DPV1_MSAC1_WRITE	Writing a data block to a DPV1 slave

Detailed information about the various blocks can be found in the following sections.

Data types

In the PROFIBUS library the following data types (structures) are defined:

DPM_COM_ERR_TYPE	Communication error
DPM_STATE_BITS_TYPE	Bits for coupler state description
STATIONSTATUS_1_TYPE	Stationstatus_1 (DP slave diagnosis according to standard)
STATIONSTATUS_2_TYPE	Stationstatus_2 (DP slave diagnosis according to standard)
STATIONSTATUS_3_TYPE	Stationstatus_3 (DP slave diagnosis according to standard)

For detailed information about the different data types please refer to the respective block description.

Overview of blocks arranged according to their call names

Used abbreviations:

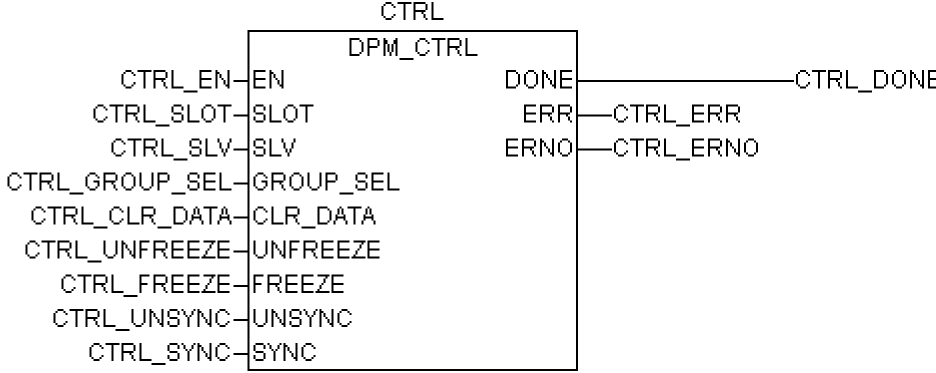
FBhv ... Function block with historical values

FBnohv ... Function block without historical values

F … Function

VE name	Type	Function
DPM_CTRL	FBhv	Sending global control commands to the DP slave
DPM_READ_INPUT	FBhv	Reading the input data of a slave which is not assigned to the master
DPM_READ_OUTPUT	FBhv	Reading the output data of a slave which is not assigned to the master
DPM_SET_PRM	FBhv	Sending user parameters to a DP slave
DPM_SLV_DIAG	FBhv	Polling of detailed diagnostic data of a DP slave
DPM_STAT	FBnohv	Reading out the status of the PROFIBUS coupler
DPM_SYS_DIAG	FBnohv	Reading out the status survey of all DP slaves
DPV1_MSAC1_READ	FBhv	Reading a data block from a DPV1 slave
DPV1_MSAC1_WRITE	FBhv	Writing a data block to a DPV1 slave

DPM_CTRL Sending global control commands to the DP Slave



The block DPM_CTRL sends global control commands used for time synchronization of process data of several DP slaves.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Profibus_AC500_V10.lib	

Block type

Function block with historical values

Parameter

Instance		DPM_CTRL	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
SLV	Input	BYTE	Selection of the called DP slaves
GROUP_SEL	Input	BYTE	Selection of the called slave groups
CLR_DATA	Input	BOOL	Resetting the output data of the DP slaves
UNFREEZE	Input	BOOL	Termination of input data synchronization
FREEZE	Input	BOOL	Synchronization of input data
UNSYNC	Input	BOOL	Termination of output data synchronization
SYNC	Input	BOOL	Synchronization of output data
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block DPM_CTRL implements the PROFIBUS function Global_Control. Global_Control is a broadcast function.

Using global control commands, the output data of one, several or all slaves can be reset and input or output data of slaves can be synchronized. The commands are selected by different combinations of the

outputs CLR_DATA, FREEZE / UNFREEZE and SYNC / UNSYNC. The called slaves are selected using three parameters. First, during project planning, the slaves can be divided into logical groups. Then, during runtime, the slaves can be called individually or in groups via the block inputs SLV and GROUP_SLV.

Every time a FALSE → TRUE edge is applied to input EN, DPM_CTRL reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

EN BOOL

If a FALSE → TRUE edge is applied to input EN, all further inputs are read in.

If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at output ERR. Additionally, the termination of the request processing is indicated by DONE = TRUE. While the request is processed, state changes at input EN are recognized but not evaluated.

SLOT BYTE

At input SLOT the coupler slot (module number) is selected which should be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

SLV BYTE

At input SLV the bus address of the DP slave is specified, to which a global control command shall be sent. With SLV = 0..126, a particular slave with the corresponding bus address is called directly, independent of the group to which it was assigned during configuration and independent of the value applied at block input GROUP_SEL. If SLV = 127 and GROUP_SEL = 0, all slaves are called simultaneously. The selection of individual slave groups is done with SLV = 127 and a combination of GROUP_SEL and the group assignment made during configuration. For that reason, the block outputs SLV and GROUP_SEL always have to be considered together with the group assignment made during configuration. The possible combinations are explained after the description of the block inputs.

GROUP_SEL BYTE

At input GROUP_SEL the slave groups are selected, to which a global control command shall be sent. With SLV = 0..126, a particular slave with the corresponding bus address is called directly, independent of the group assignment which was made during configuration and independent of the value applied at block input GROUP_SEL. If GROUP_SEL = 0 and SLV = 127, all slaves are called simultaneously. The selection of individual slave groups is done with SLV = 127 and a combination of GROUP_SEL and the group assignment made during configuration. For that reason, the block outputs SLV and GROUP_SEL always have to be considered together with the group assignment made during configuration. The possible combinations are explained after the description of the block inputs.

CLR_DATA BOOL

Using input CLR_DATA, the output data of slaves can be reset on site. The called slave resets its outputs on-site to 0 when it receives a global control command with CLR_DATA = TRUE. If the CLR_DATA part of a command is FALSE, the outputs of the called slaves keep their current state. Slaves which are not called ignore the entire command. The possible combinations of commands are explained after the description of the block inputs.

UNFREEZE BOOL

When input UNFREEZE is TRUE, the synchronization mode for input data of the called slaves is terminated, regardless of the current state of input FREEZE (TRUE or FALSE). Then, the called slaves forward their input values directly to the master again. If the UNFREEZE part of a command is FALSE, the called slaves keep their current state. The UNFREEZE command is ignored, if the called slave is not in the FREEZE state. Slaves which are not called ignore the entire command. UNFREEZE always has to be considered together with FREEZE. The possible combinations of commands are explained after the description of the block inputs.

FREEZE BOOL

When input FREEZE is TRUE and input UNFREEZE is FALSE at the same time, the called slaves change to input data synchronization mode. This mode is activated with the first FREEZE command. As a result, the called slaves simultaneously freeze the values currently applying at their local inputs and store them temporarily. During the subsequent process data cycles, the temporarily stored input values are transmitted to the master, regardless of possible changes of the input values in the mean time. When another FREEZE command is received, the temporarily stored input values are updated, i.e. the called slaves simultaneously store the present input values into an internal buffer once again and then transmit these values to the master during the subsequent cycles. FREEZE always has to be considered together with UNFREEZE. The possible combinations of commands are explained after the description of the block inputs.

UNSYNC BOOL

If input UNSYNC is TRUE, the synchronization mode for the output data of the called slaves is terminated, regardless of the current state of input SYNC (TRUE or FALSE). From now on, the called slaves immediately forward the output data received from the master to their own outputs again. If the UNSYNC part of a command is FALSE, the called slaves keep their current state. The UNSYNC command is ignored if the called slave is not in the SYNC state. Slaves which are not called ignore the entire command. UNSYNC always has to be considered together with SYNC. The possible combinations of commands are explained after the description of the block inputs.

SYNC BOOL

If input SYNC is TRUE and input UNSYNC is FALSE at the same time, the called slaves change to output data synchronization mode. This mode is activated with the first SYNC command. As a result, the called slaves freeze the current states of their local outputs. The output data sent during the subsequent process data cycles are first stored only locally in these slaves. On reception of another SYNC command, the slaves then simultaneously apply these temporarily stored values to their outputs. SYNC always has to be considered together with UNSYNC. The possible combinations of commands are explained after the description of the block inputs.

DONE BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred during the block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during the request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

Function call in IL

```
CAL  CTRL
     (EN := CTRL_EN,
     SLOT := CTRL_SLOT,
     SLV := CTRL_SLV
     GROUP_SEL :=CTRL_GROUP_SEL
     CLR_DATA := TRL_CLR_DATA,
     UNFREEZE := CTRL_UNFREEZE,
     FREEZE := CTRL_FREEZE,
     UNSYNC := CTRL_UNSYNC,
     SYNC := CTRL_SYNC)

LD   CTRL.DONE
ST   CTRL_DONE

LD   CTRL.ERR
ST   CTRL_ERR

LD   SYSDIAG.ERNO.ERR
ST   SYSDIAG_ERNO
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
CTRL
     (EN := CTRL_EN,
     SLOT := CTRL_SLOT,
     SLV := CTRL_SLV,
     GROUP_SEL := CTRL_GROUP_SEL,
     CLR_DATA := CTRL_CLR_DATA,
     UNFREEZE := CTRL_UNFREEZE,
     FREEZE := CTRL_FREEZE,
     UNSYNC := CTRL_UNSYNC,
     SYNC := CTRL_SYNC);

CTRL_DONE := CTRL.DONE;
CTRL_ERR  := CTRL.ERR;

SYSDIAG_ERNO:= SYSDIAG.ERNO;
```

Selection of the called slaves

During offline configuration stage of the master using the configuration tool SYCON.net, the master is first informed with which slaves it should exchange process data. At this point, the inserted slaves are not yet assigned to a particular group and can only be called individually using their bus address. In a further step the slaves can be combined to logical groups (refer to the SYCON.net documentation). For this purpose, the global properties of each single group must be determined first. Each group can either have only SYNC properties, or only FREEZE properties or both properties. Up to eight groups can be defined. Then, one or several groups have to be determined to which the individual slave shall be assigned (as part of the slave configuration or of the slave properties). The group assignment itself does not influence the cyclic exchange of process data. It becomes only effective in combination with global control commands.

When assigning parameters to the slaves during the boot process of the bus (this is done by the master), each slave is informed about its group assignment. This information is summarized in one byte, where each single bit represents one of the eight possible groups.

```
-----------------------
G8 G7 G6 G5 G4 G3 G2 G1
-----------------------
7  6  5  4  3  2  1  0
```

For instance, if a slave is to be assigned to the groups 7 and 1, the master sends a byte with the decimal value 65 (= binary value 01000001) to this slave.

The block inputs SLV and GROUP_SEL specify which slaves are to be called by a global control command during runtime. SLV = 0..126 calls only the slave with a bus address = SLV, independent of the group assignment and of the value applied at input GROUP_SEL. All other slaves discard this telegram. Applying SLV = 127 and GROUP_SEL = 0 calls all slaves connected to the bus, independent of their group assignment. The group assignment defined during the configuration is only considered by the slaves if they receive a global control command with SLV = 127 and GROUP_SEL unequal to 0. In this case, each slave compares the GROUP_SEL value with the group assignment byte received during parameterization. The slave accepts the command if a bit-wise collation of these two values delivers a result unequal to 0 and discards the command if the collation delivers a value of 0. For instance, if a slave which is assigned to groups 1 and 7 (see above) receives a global control command with SLV = 127 and GROUP_SEL = 64 dec. (= 01000000 bin.), the command is also (among others) addressed to this slave.

Group assignment		0	1	0	0	0	0	0	1
GROUP_SEL	AND	0	1	0	0	0	0	0	0
Result of the comparison		0	1	0	0	0	0	0	0

The following table shows the possible combinations of the three parameters SLV, GROUP_SEL and group assignment and lists the slaves called with these combinations.

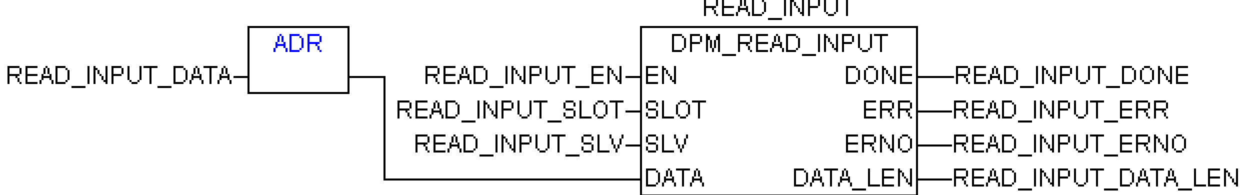
SLV	GROUP_SEL	Group assignment	Called slaves
0...126	X	X	Only slave with bus address = SLV
127	0	X	All slaves
127	1 - 255	1 - 255	Slaves, for which a bit-wise collation of group assignment and GROUP_SEL delivers a value unequal to 0.

Possible combinations of global control commands

During the process data exchange, the master cyclically transmits the output data to the corresponding slave which applies these data immediately to its outputs. In return, a slave transmits the values currently applied at its inputs to the master. During this process, the slaves are called one after the other within one bus cycle of the master. As a result, a small time difference appears between the points of time at which the individual slaves apply the received data at their local outputs. In the same way, the points of time differ at which the acquisition of values at the slave inputs and their transmission to the master takes place. A time-synchronization of the inputs or outputs is achieved with the help of global commands. While a CLR_DATA causes all called slaves to set their outputs to 0 once and at the same time, the combinations of SYNC / UNSYNC or FREEZE / UNFREEZE must be considered together. The following table shows the possible combinations within a global control command and explains their effects.

CLR_ DATA	SYNC	UN- SYNC	FREEZE	UN- FREEZE	Effect
1	X	X	X	X	All called slaves set their outputs to 0
X	0	0	X	X	No effect to SYNC mode
X	0	1	X	X	SYNC mode for output data is terminated
X	1	0	X	X	SYNC mode; the output data received last are applied to the outputs
X	1	1	X	X	SYNC mode for output data is terminated
X	X	X	0	0	No effect to FREEZE mode
X	X	X	0	1	FREEZE mode for input data is terminated
X	X	X	1	0	FREEZE mode; current input values are stored and transmitted to the master during the subsequent cycles
X	X	X	1	1	FREEZE mode for input data is terminated

DPM_READ_INPUT Reading the input data of a slave which is not assigned to the master



Using the block DPM_READ_INPUT even the input data of slaves can be read which are not assigned to the master, i.e. which were not configured by the master.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Profibus_AC500_V10.lib	

Block type

Function block with historical values

Parameter

Instance		DPM_READ_INPUT	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
SLV	Input	BYTE	DP slave address
DATA	Input	DWORD	Memory address for input data (via ADR operator)
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
DATA_LEN	Output	BYTE	Length of the read input data

Description

The block DPM_READ_INPUT implements the acyclic PROFIBUS function DDLM_Read_Input. Using this function the master can read also input data of slaves which are assigned to other masters. DPM_READ_INPUT works outside the cyclic process data exchange.

Every time a FALSE → TRUE edge is applied to input EN, DPM_READ_INPUT reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

EN BOOL

If a FALSE → TRUE edge is applied to input EN, all further inputs are read in.

If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at output ERR. Additionally, the termination of the request processing is indicated by DONE = TRUE. While the request is processed, state changes at input EN are recognized but not evaluated.

SLOT BYTE

At input SLOT the coupler slot (module number) is selected which should be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

SLV BYTE

At input SLV the bus address of the DP slave is applied, the input data of which shall be read.

DATA DWORD

At input DATA the address of the variable to be used to store the received input data is specified via the address operator ADR. The size of this variable must be big enough to store all input data of the slave (e.g. BYTE array). Furthermore, the format (BYTE, WORD, etc.) of the slave inputs must be considered. If the slave has mixed inputs of different types, it is recommended to first define a STRUCT data type which represents an image of the slave's structure (see I/O configuration of the slave) and then to declare a variable of this type.

DONE BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read via output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred during the block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during the request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

DATA_LEN BYTE

Output DATA_LEN displays the length (in bytes) of the input data read by the slave. DATA_LEN is only valid if DONE is TRUE and ERR is FALSE. If DATA_LEN contains a value X which is not 0, the block has stored X bytes of input data in the variable specified at DATA. For instance, if DATA is a byte array with start index 1, the valid input data of the slave are contained in the entries DATA[1] to DATA[X].

Function call in IL

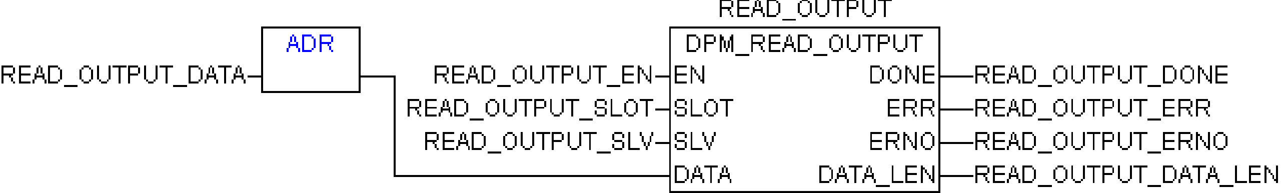
```
LD   READ_INPUT_DATA
ADR
ST   READ_INPUT_DATA_ADR
CAL  READ_INPUT
     (EN := READ_INPUT_EN,
     SLOT := READ_INPUT_SLOT,
     SLV := READ_INPUT_SLV,
     DATA := READ_INPUT_DATA,)

LD   READ_INPUT.DONE
ST   READ_INPUT_DONE

LD   READ_INPUT.ERR
ST   READ_INPUT_ERR

LD   SYSDIAG.ERNO
ST   SYSDIAG_ERNO

LD   READ_INPUT.DATA_LEN
ST   READ_INPUT_DATA_LEN
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
READ_INPUT
     (EN := RAED_INPUT_EN,
     SLOT := RAED_INPUT_SLOT,
     SLV := RAED_INPUT_SLV,
     DATA := ADR(RAED_INPUT_DATA));

READ_INPUT_DONE:= READ_INPUT.DONE;
READ_INPUT_ERR:= READ_INPUT.ERR;
SYSDIAG_ERNO:= SYSDIAG.ERNO;
READ_INPUT_DATA_LEN:= READ_INPUT.DATA_LEN;
```

DPM_READ_OUTPUT Reading the output data of a slave which is not assigned to the master



Using the block DPM_READ_OUTPUT even the output data of slaves can be read which are not assigned to the master, i.e. which were not configured by the master. Writing output data is not possible.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Profibus_AC500_V10.lib	

Block type

Function block with historical values

Parameter

Instance		DPM_READ_OUTPUT	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
SLV	Input	BYTE	Bus address of the called DP slave
DATA	Input	DWORD	Memory address for output data (via ADR operator)
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
DATA_LEN	Output	BYTE	Length of the read output data

Description

The block DPM_READ_OUTPUT implements the acyclic PROFIBUS function DDLM_Read_Output. Using this function the master can also read output data of slaves which are assigned to other masters. Write access to output data of these slaves is not possible. DPM_READ_OUTPUT works outside the cyclic process data exchange.

Every time a FALSE → TRUE edge is applied to input EN, DPM_READ_OUTPUT reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

EN BOOL

If a FALSE → TRUE edge is applied to input EN, all further inputs are read in.

If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at output ERR. Additionally, the termination of the request processing is indicated by DONE = TRUE. While the request is processed, state changes at input EN are recognized but not evaluated.

SLOT BYTE

At input SLOT the coupler slot (module number) is selected which should be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

SLV BYTE

At input SLV the bus address of the DP slave is applied, the output data of which shall be read.

DATA DWORD

At input DATA the address of the variable to be used to store the received output data is specified via the address operator ADR. The size of the variable must be big enough to store all output data of the slave (e.g. BYTE array). Furthermore, the format (BYTE, WORD, etc.) of the slave outputs must be considered. If the slave has mixed outputs of different types, it is recommended to first define a STRUCT data type which represents an image of the slave's output structure (see I/O configuration of the slave) and then to declare a variable of this type.

DONE BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred during the block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during the request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

DATA_LEN BYTE

DATA_LEN outputs the length (in bytes) of the output data read by the slave. DATA_LEN is only valid if DONE = TRUE and ERR = 0. If DATA_LEN contains a value X which is not 0, the block has stored X bytes of output data in the variable specified at DATA. For instance, if DATA is a byte array with start index 1, the valid output data of the slave are contained in the entries DATA[1] to DATA[X].

Function call in IL

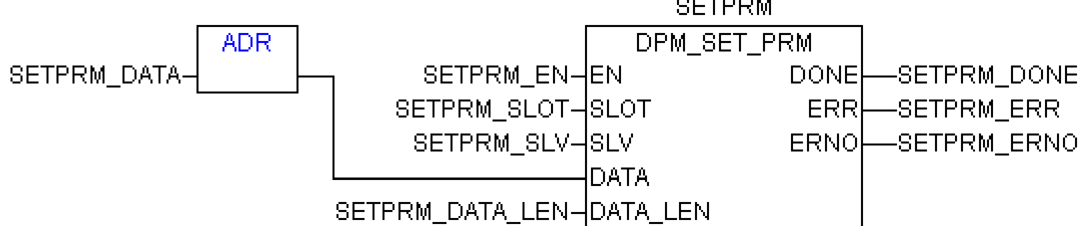
```
LD   READ_OUTPUT_DATA
ADR
ST   READ_OUTPUT_DATA_ADR
CAL  READ_OUTPUT
     (EN := READ_OUTPUT_EN,
     SLOT := READ_OUTPUT_SLOT,
     SLV := READ_OUTPUT_SLV,
     DATA := READ_OUTPUT_DATA,)

LD   READ_OUTPUT.DONE
ST   READ_OUTPUT_DONE

LD   READ_OUTPUT.ERR
ST   READ_OUTPUT_ERR

LD   SYSDIAG.ERNO
ST   SYSDIG_ERNO

LD   READ_OUTPUT.DATA_LEN
ST   READ_OUTPUT_DATA_LEN
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
READ_OUTPUT
     (EN := READ_OUTPUT_EN,
     SLOT := READ_OUTPUT_SLOT,
     SLV := READ_OUTPUT_SLV,
     DATA := ADR(READ_OUTPUT_DATA));

READ_OUTPUT_DONE:= READ_OUTPUT.DONE;
READ_OUTPUT_ERR:= READ_OUTPUT.ERR;
SYSDIAG_ERNO:= SYSDIAG.ERNO;
READ_OUTPUT_DATA_LEN:= READ_OUTPUT.DATA_LEN;
```

DPM_SET_PRM Sending user parameters to a DP slave



Using the block DPM_SET_PRM the user parameters of a slave which were preset during configuration can be modified during runtime. The new parameters are immediately sent to the corresponding slave. After re-starting the controller or the slave, the values preset in the configuration data are used again.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Profibus_AC500_V10.lib	

Block type

Function block with historical values

Parameter

Instance		DPM_SETPRM	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
SLV	Input	BYTE	Bus address of the called DP slave
DATA	Input	DWORD	Memory address of the parameters (via ADR operator)
DATA_LEN	Input	BYTE	Length of parameter data to be sent (byte value)
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block DPM_SET_PRM implements the PROFIBUS function Set_Prm.

With the function Set_Prm the user parameters of a slave can be modified during runtime. The parameters are immediately sent to the slave. They are valid until modified parameters are transmitted once again. After re-starting the controller or the slave, the values preset in the configuration data are used again.

Format and length of the user parameters are slave-specific. Due to this, the block DPM_SET_PRM provides inputs where only the variable address and the length of the user parameters to be sent must be specified. It is in the responsibility of the user that the data comply with the requirements of the corresponding device regarding format and length (e.g. by defining a structure).

Every time a FALSE → TRUE edge is applied to input EN, DPM_SET_PRM reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

EN BOOL

If a FALSE → TRUE edge is applied to input EN, all further inputs are read in.

If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at output ERR. Additionally, the termination of the request processing is indicated by DONE = TRUE. While the request is processed, state changes at input EN are recognized but not evaluated.

SLOT BYTE

At input SLOT the coupler slot (module number) is selected which should be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

SLV BYTE

At input SLV the bus address of the DP slave is specified, to which the user parameters shall be sent. Valid addresses are values between 0 and 126.

DATA DWORD

At input DATA the address of the variable from which onwards the parameters to be sent are stored is specified via the address operator ADR. The data format and the length of the variable must correspond to the structure of the user parameters of the slave. It is recommended to first define a STRUCT data type which represents an image of the slave's structure and then to declare a variable of this type. Please note that only the user parameters are to be specified at this point. The standard parameters cannot be modified during runtime. The standard parameter settings specified during configuration are automatically completed by the block.

DATA_LEN BYTE

The input DATA_LEN informs the block about the length (number of bytes) of the user parameters to be transmitted. The maximum length is 237 bytes.

DONE BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred during the block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during the request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

Function call in IL

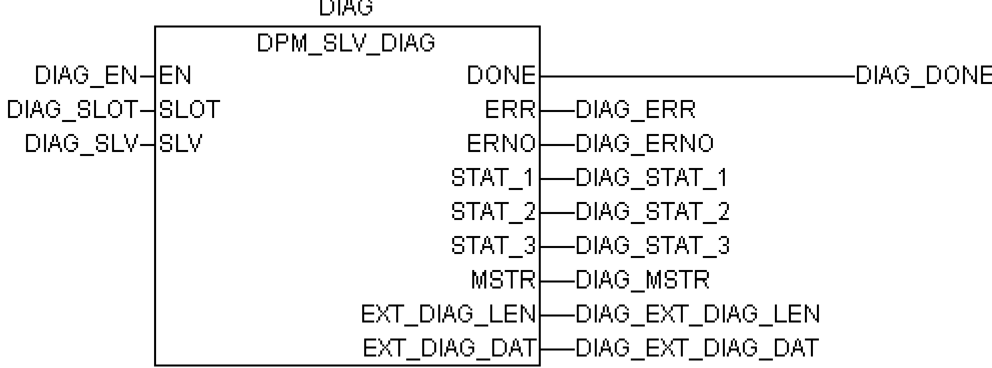
```
LD   SETPRM_DATA
ADR
ST   SETPRM_DATA_ADR
CAL  SETPRM
      (EN := SETPRM_EN,
     SLOT := SETPRM_SLOT,
     SLV := SETPRM_SLV,
     DATA := SETPRM_DATA,
     DATA_LEN := SETPRM_DATA_LEN)

LD   SETPRM.DONE
ST   SETPRM_DONE

LD   SETPRM.ERR
ST   SETPRM_ERR

LD   SYSDIAG.ERNO
ST   SYSDIAG_ERNO
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
SETPRM
       (EN := SETPRM_EN,
      SLOT := SETPRM_SLOT,
      SLV := SETPRM_SLV,
      DATA := SETPRM_DATA,
      DATA_LEN := SETPRM_DATA_LEN);

SETPRM_DONE :=  SETPRM.DONE;
SETPRM_ERR  :=  SETPRM.ERR;

SYSDIAG_ERNO:= SYSDIAG.ERNO;
```

DPM_SLV_DIAG Polling detailed diagnostic data of a DP slave



The block DPM_SLV_DIAG reads detailed diagnostic data from a DP slave.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Profibus_AC500_V10.lib	

Block type

Function block with historical values

Parameter

Instance		DPM_SLVDIAG	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
SLV	Input	BYTE	Bus address of the specific DP slave
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
STAT_1	Output	STATIONSTATUS_1_TYPE	Stationstatus_1 according to standard
STAT_2	Output	STATIONSTATUS_2_TYPE	Stationstatus_2 according to standard
STAT_3	Output	STATIONSTATUS_3_TYPE	Stationstatus_3 according to standard
MSTR	Output	BYTE	Bus address of the related DP master
EXT_DIAG_LEN	Output	BYTE	Length of the extended diagnostic data
EXT_DIAG_DAT	Output	ARRAY[1.0,238] OF BYTE	Extended diagnostic data according to standard

Description

The block DPM_SLV_DIAG implements the PROFIBUS function Slave_Diag.

Every time a FALSE → TRUE edge is applied to input EN, DPM_SLV_DIAG reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

EN BOOL

If a FALSE → TRUE edge is applied to input EN, all further inputs are read in.

If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at output ERR. Additionally, the termination of the request processing is indicated by DONE = TRUE.

While the request is processed, state changes at input EN are recognized but not evaluated.

SLOT BYTE

At input SLOT the coupler slot (module number) is selected which should be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

SLV BYTE

At input SLV the bus address of the DP slave is specified, for which diagnostic data are requested.

DONE BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred during the block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during the request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

STAT_1 STATIONSTATUS_1_TYPE

STAT_1 outputs the first octet of the DP slave diagnostic data. STAT_1 is only valid if DONE = TRUE and ERR = FALSE.

The structure of the type STATIONSTATUS_1_TYPE corresponds to the octet Stationstatus_1 defined in the standard as described below.

STAT_2 STATIONSTATUS_2_TYPE

STAT_2 outputs the second octet of the DP slave diagnostic data. STAT_2 is only valid if DONE = TRUE and ERR = FALSE.

The structure of the type STATIONSTATUS_2_TYPE corresponds to the octet Stationstatus_2 defined in the standard as described below.

STAT_3 STATIONSTATUS_3_TYPE

STAT_3 outputs the third octet of the DP slave diagnostic data. STAT_3 is only valid if DONE = TRUE and ERR = FALSE.

The structure of the type STATIONSTATUS_3_TYPE corresponds to the octet Stationstatus_3 defined in the standard as described below.

MSTR BYTE

MSTR outputs the bus address of the DP master to which the DP slave is assigned. MSTR is only valid if DONE = TRUE and ERR = FALSE.

Using the block DPM_SLV_DIAG, diagnostic data for all DP slaves can be polled which are connected to the bus. If the diagnostic data are requested by a DP slave which was assigned to the PLC during configuration, the PLC bus address is output at MSTR. In multi-master systems it is possible that diagnostic data are also requested by DP slaves which are assigned to other DP masters. In this case, MSTR outputs the bus address of the DP master to which the requesting DP slave is assigned.

EXT_DIAG_LEN BYTE

EXT_DIAG_LEN outputs the number of valid bytes, following in EXT_DIAG_DAT. If EXT_DIAG_LEN = 0, no extended diagnostic data are available. Otherwise, EXT_DIAG_DAT[1] to EXT_DIAG_DAT[EXT_DIAG_LEN] contain the extended diagnostic data reported by the DP slave. EXT_DIAG_LEN is only valid if DONE = TRUE and ERR = FALSE.

EXT_DIAG_DAT ARRAY [1..238] OF BYTE

At output EXT_DIAG_DAT the extended diagnostic data reported by the DP slave are applied as a byte array. The data in EXT_DIAG_DAT are only valid if DONE = TRUE, ERR = FALSE and EXT_DIAG_LEN > 0.

The extended diagnostic data are structured according to the standard. Since the meaning of these data strongly depends on the used devices, no automatic interpretation by the block is possible at this point. The evaluation of the data must be performed by the user, with the aid of the particular device description and the GSD file respectively. The general structure of the extended diagnostic data (according to the standard) is described below.

Function call in IL

```
CAL   DIAG
      (EN   := DIAG_EN,
      SLOT  := DIAG_SLOT,
      SLV   := DIAG_SLV)

LD    DIAG.DONE
ST    DIAG_DONE

LD    DIAG.ERNO
ST    DIAG_ERNO

LD    DIAG.ERR
ST    DIAG_ERR

LD    DIAG.STAT_1
ST    DIAG_STAT_1

LD    DIAG.STAT_2
ST    DIAG_STAT_2

LD    DIAG.STAT_3
ST    DIAG_STAT_3

LD    DIAG.MSTR
ST    DIAG_MSTR

LD    DIAG.EXT_DIAG_LEN
ST    DIAG_EXT_DIAG_LEN

LD    DIAG.EXT_DIAG_DAT
ST    DIAG_EXT_DIAG_DAT
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
DIAG
     (EN            := DIAG_EN,
     SLOT           := DIAG_SLOT,
     SLV            := DIAG_SLV);

DIAG_DONE          := DIAG.DONE;
DIAG_ERR           := DIAG.ERR;
DIAG_ERNO          := DIAG.ERNO;
DIAG_STAT_1        := DIAG.STAT_1;
DIAG_STAT_2        := DIAG.STAT_2;
DIAG_STAT_3        := DIAG.STAT_3;
DIAG_MSTR          := DIAG.MSTR;
DIAG_EXT_DIAG_LEN := DIAG.EXT_DIAG_LEN;
DIAG_EXT_DIAG_DAT := DIAG.EXT_DIAG_DAT;
```

Structure of DP slave diagnostic data

The block DPM_SLV_DIAG divides the diagnostic data of a DP slave into sections and applies them to the corresponding outputs. The structure of the DP slave diagnostic data is prescribed by the standard as follows:

Octet 1	Stationstatus_1	STAT_1
Octet 2	Stationstatus_2	STAT_2
Octet 3	Stationstatus_3	STAT_3
Octet 4	Master_Add	MSTR
Octet 5 to 6	Ident_Number	
Octet 7 to 244	Ext_Diag_Data	EXT_DIAG_DAT[1] to EXT_DIAG_DAT[EXT_DIAG_LEN]

Stationstatus_1   STAT_1   STATIONSTATUS_1_TYPE

The diagnostic byte Stationstatus_1 (defined within the standard) is output at STAT_1 of the block DPM_SLV_DIAG. It is represented as a structure of the data type STATIONSTATUS_1_TYPE. Within the PROFIBUS library the structure STATIONSTATUS_1_TYPE is declared as follows:

```
TYPE STATIONSTATUS_1_TYPE:
STRUCT
    NON_EXISTENT:     BOOL;
    NOT_READY:        BOOL;
    CFG_FAULT:        BOOL;
    EXT_DIAG:         BOOL;
    NOT_SUPPORTED:    BOOL;
    INVALID_RESPONSE: BOOL;
    PRM_FAULT:        BOOL;
    MASTER_LOCK:      BOOL;
END_STRUCT
END_TYPE
```

NON_EXISTENT   BOOL

If this bit is set, the PROFIBUS coupler has not found a DP slave at the bus address applied at block input SLV.

NOT_READY   BOOL

This bit is set to TRUE by the DP slave if the DP slave is not ready for I/O data exchange.

CFG_FAULT   BOOL

This bit is set to TRUE by the DP slave if the configuration data (nominal configuration) received from the DP master do not match the data stored in the DP slave (actual configuration).

EXT_DIAG   BOOL

If this bit is TRUE, extended diagnostic data are available in EXT_DIAG_DAT. If this bit is not set (FALSE) and EXT_DIAG_LEN > 0, possibly a status message is available in EXT_DIAG_DAT. The meaning of such a status message is device-dependent.

NOT_SUPPORTED   BOOL

This bit is set to TRUE by the DP slave if an unsupported function was requested before.

INVALID_RESPONSE   BOOL

If this bit is TRUE, the PROFIBUS coupler has not received a plausible response from the requested DP slave.

PRM_FAULT BOOL

The DP slave sets this bit to TRUE if the parameter data received last were faulty.

MASTER_LOCK BOOL

This bit is set to TRUE if the DP slave is assigned to another DP master. In this case, MSTR contains the bus address of this DP master.

Stationstatus_2 STAT_2 STATIONSTATUS_2_TYPE

The diagnostic byte Stationstatus_2 (defined within the standard) is output at STAT_2 of the block DPM_SLV_DIAG. It is represented as a structure of the data type STATIONSTATUS_2_TYPE. The structure STATIONSTATUS_2_TYPE is declared as follows in the PROFIBUS library:

```
TYPE STATIONSTATUS_2_TYPE:
STRUCT
     PRM_REQ:           BOOL;
     STAT_DIAG:         BOOL;
     DP_Slave:          BOOL;
     WD_ON:             BOOL;
     FREEZE_MODE:       BOOL;
     SYNC_MODE:         BOOL;
     reserved:          BOOL;
     DEACTIVATED:       BOOL;
END_STRUCT
END_TYPE
```

PRM_REQ BOOL

This bit is set to TRUE by the DP slave, if reparameterization and reconfiguration of the slave is required (e.g. when adding an additional I/O module). The bit remains set until reparameterization is done.

STAT_DIAG BOOL

This bit is set to TRUE by the DP slave if the slave has a static diagnosis. A DP slave with static diagnosis is not ready for I/O data exchange.

DP_Slave BOOL

This bit is permanently set to TRUE.

WD_ON BOOL

This bit is set to TRUE by the DP slave if the slave response monitoring is active.

FREEZE_MODE BOOL

This bit is set to TRUE by the DP slave if the slave is currently running in Freeze mode.

SYNC_MODE BOOL

This bit is set to TRUE by the DP slave if the slave is currently running in Sync mode.

reserved BOOL

This bit is reserved and currently not used.

DEACTIVATED BOOL

This bit is set to TRUE if the DP slave is marked as non-active in the DP master's configuration data and has been taken out of the cyclic I/O data exchange.

Stationstatus_3   STAT_3   STATIONSTATUS_3_TYPE

The diagnostic byte Stationstatus_3 (defined within the standard) is output at STAT_3 of the block DPM_SLV_DIAG. It is represented as a structure of the data type STATIONSTATUS_3_TYPE. The structure STATIONSTATUS_3_TYPE is declared as follows within the PROFIBUS library:

```
TYPE STATIONSTATUS_3_TYPE:
STRUCT
    reserved0:          BOOL;
    reserved1:          BOOL;
    reserved2:          BOOL;
    reserved3:          BOOL;
    reserved4:          BOOL;
    reserved5:          BOOL;
    reserved6:          BOOL;
    EXT_DIAG_OVERFLOW:  BOOL;
END_STRUCT
END_TYPE
```

Master_Add   MSTR   BYTE

The DP slave enters the DP master's address into this octet by which it was parameterized (i.e. to which it is assigned). If the DP slave was not yet parameterized by any DP master, MSTR is set to 255.

Ident_Number

Here, the DP slave enters its identification number in the diagnosis telegram. The identification number is assigned by the PNO (PROFIBUS Nutzerorganisation e.V. = PROFIBUS user organization) for each device type. This number is a firm component of the GSD file. The block DPM_SLV_DIAG does not output the device identification number, because the number is not necessary for evaluating the diagnostic data.

Ext_Diag_Data EXT_DIAG_DAT   ARRAY [1..238] OF BYTE

The six bytes of standard diagnostic data described above have to be supported by each DP slave. Optionally, a DP slave can additionally provide extended diagnostic data. This is the case if a value greater than 6 is assigned to the item Max_Diag_Data_Len in the GSD file of the DP slave. The format of the extended diagnosis is defined by the standard. Since the extended diagnostic data are not static on the one side and can contain manufacturer-specific entries on the other side, no automatic data evaluation can be performed by the block DM_SLV_DIAG.

The evaluation of extended diagnostic data must be performed by the user with the aid of the GSD file for the particular DP slave and the description of the general data format given below.

The extended diagnostic data are divided into three parts (sections):

- device-related diagnosis

- module-related diagnosis

- channel-related diagnosis

Device-related diagnosis

The device-related diagnosis section contains general diagnostic information, such as overtemperature or undervoltage. This section starts with a header byte, the highest two bits of which are permanently set to 00. The lower six bits indicate the length of the following block, including the header byte itself.

0	0	L	e	n	g	t	h
7	6	5	4	3	2	1	0

The device-related diagnostic data are defined by the manufacturer. For detailed information about their meaning please refer to the particular device documentation.

Module-related diagnosis

The module-related diagnosis section contains diagnostic information which can be assigned directly to the particular I/O modules of the device. This section containing the module-related diagnosis starts with a header byte, the highest two bits of which are permanently set to 01. The lower six bits indicate the length of the following block, including the header byte itself.

0	1	L	e	n	g	t	h
7	6	5	4	3	2	1	0

In the following block, one single bit is assigned to each module. The module index is represented by the bit offset within the block (please refer to the example). A bit which is set to TRUE means that diagnosis is required for the related I/O module.

Channel-related diagnosis

In the channel-related diagnosis section, the diagnosed channels and the cause for the diagnosis are entered in sequence. Each entry consists of three bytes and starts with a header byte, the highest two bits of which are permanently set to 10. The lower six bits contain the index of the module for which the following diagnosis was made.

1	0	M	o	d	u	l	e
7	6	5	4	3	2	1	0

The lower six bits of the following byte contain the number of the channel which reports a diagnosis. The highest two bits indicate whether the specific channel is an input channel, an output channel or an I/O channel.

I	O	C	h	a	n	n	l
7	6	5	4	3	2	1	0

The direction identifier in the bits 6 and 7 is encoded as follows:

```
0  0  Reserved
0  1  Input
1  0  Output
1  1  Input/Output
```

The third byte of each entry contains the channel type in its upper three bits and the error type in the lower five bits.

C	h	n	E	r	r	o	r
7	6	5	4	3	2	1	0

The channel type (channel width) is encoded as follows:

```
0  0  0   Reserved
0  0  1   Bit
0  1  0   2 Bit
0  1  1   4 Bit
1  0  0   Byte
1  0  1   Word
1  1  0   2 Word
1  1  1   Reserved
```

The encoding of the error type is as follows:

0	Reserved	0	0	0	0	0
1	Short circuit	0	0	0	0	1
2	Undervoltage	0	0	0	1	0
3	Overvoltage	0	0	0	1	1
4	Overload	0	0	1	0	0
5	Overtemperature	0	0	1	0	1
6	Cable brake	0	0	1	1	0
7	Upper limit exceeded	0	0	1	1	1
8	Lower limit exceeded	0	1	0	0	0
9	Error	0	1	0	0	1
10	Reserved	0	1	0	1	0
:	:	0	1	x	x	x
15	Reserved	0	1	1	1	1
16	Manufacturer-specific	1	0	0	0	0
:	:	1	x	x	x	x
31	Manufacturer-specific	1	1	1	1	1

The valid length of the complete extended diagnostic data is indicated at output EXT_DIAG_LEN of the block DPM_SLV_DIAG. When evaluating the diagnosis, only data have to be considered which are contained in the range EXT_DIAG_DAT[1] to EXT_DIAG_DAT[EXT_DIAG_LEN].

Example for extended diagnostic data

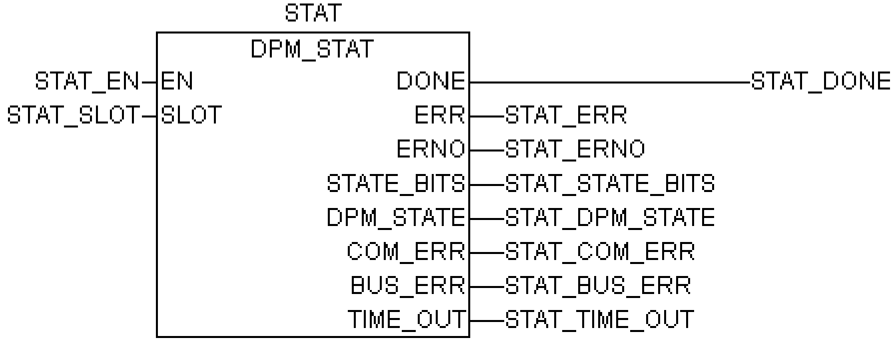
EXT_DIAG_LEN = 15

	7	6	5	4	3	2	1	0	
EXT_DIAG_DAT[1]	**0**	**0**	0	0	0	1	0	0	Device-related diagnosis; Length: 4 bytes incl. header byte
EXT_DIAG_DAT[2]	X	x	x	X	X	x	X	X	Device-related diagnosis
EXT_DIAG_DAT[3]	X	x	x	X	X	x	X	X	Length: 3 bytes
EXT_DIAG_DAT[4]	X	x	x	X	X	x	X	X	Meaning of the data is manufacturer-specific
EXT_DIAG_DAT[5]	**0**	**1**	0	0	0	1	0	1	ID-related diagnosis; Length: 5 bytes incl. header byte
EXT_DIAG_DAT[6]	0_{7}	0_{6}	0_{5}	0_{4}	0_{3}	0_{2}	0_{1}	1_{0}	Module 0 with diagnosis
EXT_DIAG_DAT[7]	0_{15}	0_{14}	0_{13}	1_{12}	0_{11}	0_{10}	0_{9}	0_{8}	Module 12 with diagnosis
EXT_DIAG_DAT[8]	0_{23}	0_{22}	0_{21}	0_{20}	0_{19}	1_{18}	0_{17}	0_{16}	Module 18 with diagnosis
EXT_DIAG_DAT[9]	0_{31}	1_{30}	0_{29}	0_{28}	0_{27}	0_{26}	0_{25}	0_{24}	Module 30 with diagnosis
EXT_DIAG_DAT[10]	**1**	**0**	0	0	0	0	0	0	Channel-related diagnosis module 0
EXT_DIAG_DAT[11]	0	1	0	0	0	0	1	0	Channel 2, Input
EXT_DIAG_DAT[12]	0	0	1	0	0	1	0	0	Overload, channel organized bit-wise
EXT_DIAG_DAT[13]	**1**	**0**	0	0	1	1	0	0	Channel-related diagnosis module 12
EXT_DIAG_DAT[14]	1	0	0	0	0	1	1	0	Channel 6, Output
EXT_DIAG_DAT[15]	1	0	1	0	0	1	1	1	Upper limit exceeded, channel organized word-wise

Example program for evaluating extended diagnostic data

A detailed example program for the evaluation of extended diagnostic data can be found on the CD Control Builder AC500 and in the programming system online help.

DPM_STAT Reading out the status of the PROFIBUS coupler



DPM_STAT outputs the PROFIBUS coupler status. The outputs provide information about the communication state and error events.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Profibus_AC500_V10.lib	

Block type

Function block without historical values

Parameter

Instance		DPM_STAT	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
STAT_BITS	Output	DPM_STATE_BITS_TYPE	Atypical communication states
DPM_STATE	Output	WORD	DP master state according to standard
COM_ERR	Output	DPM_COM_ERR_TYPE	Communication error
BUS_ERR	Output	WORD	Number of serious bus errors
TIME_OUT	Output	WORD	Number of timeouts

Description

The block DPM_STAT outputs the current PROFIBUS coupler status. DPM_STAT is active if input EN = TRUE. While the block is active, the current values are permanently displayed at the outputs.

EN BOOL

The block can be activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are applied to the outputs.

SLOT BYTE

At input SLOT the coupler slot (module number) is selected which should be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred during the block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during the request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

STATE_BITS DPM_STATE_BITS_TYPE

STATE_BITS outputs atypical communication states at the PROFIBUS. STAT_BITS is only valid if EN = TRUE and ERR = FALSE.

The structure of the type DPM_STATE_BITS_TYPE is defined in the PROFIBUS library as described below.

DPM_STATE BYTE

At output DPM_STATE, the DP master state is output according to the standard. The following states are defined:

```
OFFLINE 00HEX / 00DEC
STOP    40HEX / 64DEC
CLEAR   80HEX / 128DEC
OPERATE C0HEX / 192DEC
DPM_STATE = OFFLINE
```

The PROFIBUS coupler is in initialization state if DPM_STATE has the value OFFLINE. After the initialization phase is completed, the coupler changes to STOP state.

DPM_STATE = STOP

The coupler is completely initialized if DPM_STATE has the value STOP. In this state the coupler is ready to receive configuration data. No data are exchanged with the DP slaves. The coupler has this state if no user program is running.

DPM_STATE = CLEAR

When starting the user program, the coupler changes from STOP into CLEAR state and begins, via the bus, to parameterize (set into operation) the DP slaves assigned during configuration. When the setup has been completed successfully, the coupler moves to OPERATE state. If an error occurs during parameterization, the coupler changes back to STOP state.

DPM_STATE = OPERATE

Normally, the coupler is in OPERATE state while a user program is running. In this state, the DP master exchanges I/O data with the DP slaves. If an error occurs during this process and if 'Auto Clear Mode' was selected during configuration, the coupler changes back to CLEAR state and tries to parameterize the DP slaves again. If 'Auto Clear Mode' was not selected, the coupler remains (in case of an error) in OPERATE state. When stopping the user program, the coupler also changes back to STOP state.

DPM_STATE is only valid if EN = TRUE and ERR = FALSE.

COM_ERR DPM_COM_ERR_TYPE

Output COM_ERR outputs possible communication errors. COM_ERR is only valid if EN = TRUE and ERR = FALSE.

The structure of the type DPM_COM_ERR_TYPE is defined in the PROFIBUS library and described below together with the possible errors.

BUS_ERR WORD

BUS_ERR outputs the number of serious bus errors since system startup, such as transmission line short circuits. BUS_ERR is only valid if EN = TRUE and ERR = FALSE.

TIME_OUT WORD

TIME_OUT outputs the number of timeout errors since system startup. A timeout error occurs if a DP slave does not respond to a DP master's request within the configured time. TIME_OUT is only valid if EN = TRUE and ERR = FALSE.

Function call in IL

```
CAL   STAT
(EN   :=   STAT_EN,
SLOT  :=   STAT_SLOT)

LD    STAT.DONE
ST    STAT_DONE

LD    STAT.ERR
ST    STAT_ERR

LD    STAT.ERNO
ST    STAT_ERNO

LD    STAT.STATE_BITS
ST    STAT_STATE_BITS

LD    STAT.DPM_STATE
ST    STAT_DPM_STATE

LD    STAT.COM_ERR
ST    STAT_COM_ERR

LD    STAT.BUS_ERR
ST    STAT_BUS_ERR

LD    STAT.TIME_OUT
ST    STAT_TIME_OUT
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
STAT
    (EN := STAT_EN,
   SLOT := STAT_SLOT);

STAT_DONE        := STAT.DONE;
STAT_ERR         := STAT.ERR;
STAT_ERNO        := STAT.ERNO;
STAT_STATE_BITS  := STAT.STATE_BITS;
STAT_DPM_STATE   := STAT.DPM_STATE;
STAT_COM_ERR     := STAT.COM_ERR;
STAT_BUS_ERR     := STAT.BUS_ERR;
STAT_TIME_OUT    := STAT.TIME_OUT;
```

STATE_BITS   DPM_STATE_BITS_TYPE

The structure STATE_BITS includes four Boolean variables which display different communication states. Within the PROFIBUS library, the data type DPM_STATE_BITS_TYPE is declared as follows:

```
TYPE DPM_STATE_BITS_TYPE:
STRUCT
    CTRL:       BOOL;
    AUTO_CLR:   BOOL;
    NO_EXCH:    BOOL;
    FATAL:      BOOL;
    EVENT:      BOOL;
    TIMEOUT:    BOOL;
END_STRUCT
END_TYPE
```

CTRL   BOOL

If this bit is TRUE, a parameter setting error occurred. During normal operation, CTRL should be FALSE. If this is not the case, the parameter and configuration data have to be checked.

AUTO_CLR   BOOL

This bit is only valid if ‘Auto Clear Mode’ was set during the configuration. If AUTO_CLR is TRUE, an error occurred during communication with at least one DP slave. As a result, the coupler stopped the data exchange with all DP slaves and changed back to CLEAR state (see DPM_STATE).

NO_EXCH   BOOL

This bit is set to TRUE if process data exchange with one or several DP slaves is not possible. The error can be caused by the configuration data or by the DP slaves.

FATAL   BOOL

If FATAL is TRUE, no communication via the PROFIBUS is possible due to a serious bus error (e.g. bus line short circuit). In this case, all bus lines have to be checked.

EVENT   BOOL

This bit is set to TRUE if a bus short circuit was detected. The number of detected short circuits can be read at the BUS_ERR output. After setting the bit once, its state remains TRUE.

TIMEOUT   BOOL

This bit is set to TRUE if a telegram timeout was detected. The number of detected timeouts can be read at the TIME_OUT output. After setting the bit once, its state remains TRUE.

COM_ERR   DPM_COM_ERR_TYPE

Communication errors can be located more detailed using COM_ERR. COM_ERR outputs a structure of the type DPM_COM_ERR_TYPE. This data type is declared as follows within the PROFIBUS library:

```
TYPE DPM_COM_ERR_TYPE:
STRUCT
    ADDRESS:  BYTE;
    EVENT:    BYTE;
END_STRUCT
END_TYPE
```

ADDRESS   BYTE

If an error occurs, ADDRESS contains the bus address of the faulty device (0 to 125). If ADDRESS has the value 255, the error is located in the coupler itself.

EVENT BYTE

EVENT displays the cause of an error. The following tables show the encoding of the various errors.

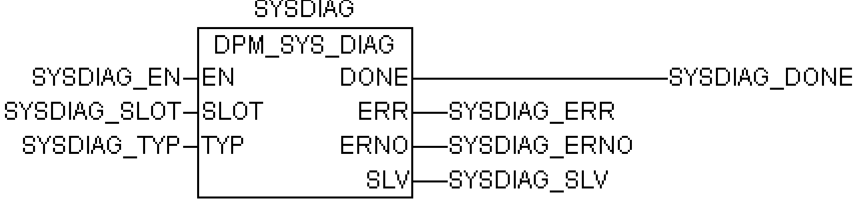
ADDRESS = 255 / Coupler error

Event	Meaning	Error source	Remedy
50	USR_INTF task not found	Coupler	Contact ABB
51	No global data field	Coupler	Contact ABB
52	FDL task not found	Coupler	Contact ABB
53	PLC task not found	Coupler	Contact ABB
54	No master parameter record	Configuration	Generate configuration in the project and reload program to controller
55	Faulty value in master parameter record	Configuration	Check configuration data for coupler in the project and/or reload program to controller
56	No slave parameter records	Configuration	Add DP slaves to configuration data and reload program to controller
57	Faulty value in a slave parameter record	Configuration	Check configuration data of subordinate DP slaves in the project and/or reload program to controller
58	Doubled slave address	Configuration	Check configuration data of subordinate DP slaves in the project for doubled bus addresses and/or reload program to controller
59	Invalid offset address output data	Configuration	Check configuration data of subordinate DP slaves in the project for invalid IEC addresses and/or reload program to controller
60	Invalid offset address input data	Configuration	Check configuration data of subordinate DP slaves in the project for invalid IEC addresses and/or reload program to controller
61	Range overlapping in output data	Configuration	Check configuration data of subordinate DP slaves in the project for overlapping IEC address ranges and/or reload program to controller
62	Range overlapping in input data	Configuration	Check configuration data of subordinate DP slaves in the project for overlapping IEC address ranges and/or reload program to controller
63	Unknown process data handshake	Controller	Supply voltage OFF/ON, otherwise contact ABB
64	Insufficient memory	Coupler	Contact ABB
65	Faulty slave parameter record	Configuration	Check configuration data of subordinate DP slaves in the project and/or reload program to controller
202	No segment available	Coupler	Contact ABB
212	Error while reading database	Configuration/ Coupler	Reload program with configuration data to controller
213	Faulty transfer structure operating system	Coupler	Contact ABB

ADDRESS = 0..125 / Error at subscriber with bus address ADDRESS

Event	Meaning	Error source	Remedy
2	Subscriber reports overflow	DP master telegram	Check configuration data of subordinate DP slave in the project and/or reload program to controller
3	Subscriber does not support requested function	DP master telegram	Check DP slave for conformity according to PROFIBUS standard
9	No data in response telegram	DP slave	Compare configuration data of subordinate DP slave in the project with actual configuration and reload program to controller if necessary
17	Subscriber does not response	DP slave	Check bus line and DP slave bus address
18	DP master not in logical token ring	DP master	Check the configured DP master bus address, the highest station address (HSA) in the other system DP masters and/or bus line for short circuits
21	Faulty parameter in request telegram	DP master telegram	Contact ABB

DPM_SYS_DIAG Reading out a status overview of all DP slaves



The block DPM_SYS_DIAG outputs a bit field at output SLV representing a state survey of all DP slaves. Three different survey types can be selected via input TYP.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Profibus_AC500_V10.lib	

Block type

Function block without historical values

Parameter

Instance		DPM_SYSDIAG	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
TYP	Input	BYTE	Selection of the survey type
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
SLV	Output	ARRAY [0..127] OF BOOL	DP slaves status survey

Description

The block DPM_SYS_DIAG outputs different surveys reporting the status of all DP slaves. Three survey types can be selected:

- configuration survey
- I/O data exchange survey
- diagnosis survey

EN BOOL

The block can be activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are applied to the outputs.

SLOT BYTE

At input SLOT the coupler slot (module number) is selected which should be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

TYP BYTE

Input TYP is used to select the status survey type to be output at SLV.

TYP = 1 configuration survey

Output SLV indicates which DP slaves have been configured successfully (i.e. set into operation) by the DP master. Please note that the DP master only sets DP slaves into operation which were assigned to the master when generating the configuration data.

TYP = 2 data exchange survey

SLV outputs all DP slaves with which the DP master exchanges data. The data exchange can only be performed with DP slaves which were configured by the DP master itself. The data exchange survey can only be requested, if the coupler is in OPERATE state.

TYP = 3 diagnosis survey

Output SLV indicates all DP slaves which have signalized an available diagnosis. The diagnosis survey can only be requested if the coupler is in OPERATE state.

DONE BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred during the block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during the request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

SLV ARRAY [0..127] OF BOOL

SLV outputs the status survey as a bit field. Every bit within this field represents a DP slave. The index corresponds to the DP slave bus address. When a bit is set to TRUE, the state selected via TYP is applicable for the corresponding DP slave.

If e.g. TYP = 1 is selected and SLV[2] = TRUE, the DP slave was successfully configured with bus address 2 by the DP master. If SLV[2] = FALSE, the configuration of the specific DP slave has not yet been completed or the DP slave is not part of the DP master configuration data.

If TYP = 2 was selected and SLV[2] = TRUE, the DP master exchanges I/O data with the DP slave having bus address 2. If SLV[2] = FALSE, the DP master currently does not exchange I/O data with the DP slave. The DP master is only able to exchange data with DP slaves which it has successfully set into operation before.

If TYP = 3, SLV[2] = TRUE means that the DP slave with bus address 2 has signalized a diagnosis. The detailed diagnosis can then be requested using the block DPM_SLVDIAG.

The bit field output at SLV is only valid if EN = TRUE and ERR = FALSE.

Function call in IL

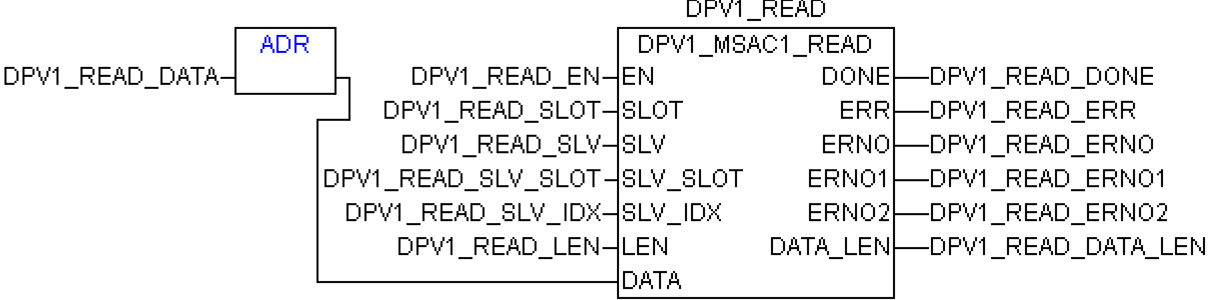
```
CAL   SYSDIAG
      (EN := SYSDIAG_EN,
      SLOT := SYSDIAG_SLOT,
      TYP := SYSDIAG_TYP)

LD    SYSDIAG.DONE
ST    SYSDIAG_DONE

LD    SYSDIAG.ERR
ST    SYSDIAG_ERR

LD    SYSDIAG.ERNO
ST    SYSDIAG_ERNO

LD    SYSDIAG.SLV
ST    SYSDIAG_SLV
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
SYSDIAG
       (EN := SYSDIAG_EN,
       SLOT := SYSDIAG_SLOT,
       TYP := SYSDIAG_TYP);

SYSDIAG_DONE:= SYSDIAG.DONE;
SYSDIAG_ERR := SYSDIAG.ERR;
SYSDIAG_ERNO:= SYSDIAG.ERNO;
SYSDIAG_SLV := SYSDIAG.SLV;
```

DPV1_MSAC1_READ Reading a data block from a DPV1 slave



The block DPV1_MSAC1_READ can be used to read a data block from a DPV1 slave by specifying the slot and index.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Profibus_AC500_V10.lib	

Block type

Function block with historical values

Parameter

Instance		DPV1_MSAC1_READ	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
SLV	Input	BYTE	DPV1 slave address
SLV_SLOT	Input	BYTE	Slot number of the data block to be read
SLV_IDX	Input	BYTE	Index of the data block to be read
LEN	Input	BYTE	Length of the data block to be read
DATA	Input	DWORD	Memory address for data block (via ADR operator)
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
ERNO1	Output	BYTE	Additional error information
ERNO2	Output	BYTE	Additional error information
DATA_LEN	Output	BYTE	Actual length of the read data

Description

The block DPV1_MSAC1_READ implements the acyclic PROFIBUS DPV1 service MSAC1_READ. Using this function, the master has read access to slot and index-related data of slaves supporting DPV1. DPV1_MSAC1_READ works outside the cyclic process data exchange.

Every time a FALSE → TRUE edge is applied to input EN, DPV1_MSAC1_READ reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

EN BOOL

If a FALSE → TRUE edge is applied to input EN, all further inputs are read in.

If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at output ERR. Additionally, the termination of the request processing is indicated by DONE = TRUE. While the request is processed, state changes at input EN are recognized but not evaluated.

SLOT BYTE

At input SLOT the coupler slot (module number) is selected which should be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

SLV BYTE

At input SLV the bus address of the DP slave is applied, the data of which shall be read. Valid values: 0..126.

SLV_SLOT BYTE

At input SLV_SLOT the number of the slot within the slave is specified, the data of which shall be read. Valid values: 0..254.

SLV_IDX BYTE

At input SLV_IDX the number of the index within the slot is specified, the data of which shall be read. Valid values: 0..254.

LEN BYTE

The length of the data block to be read is specified at input LEN. Valid values: 0..240.

DATA DWORD

At input DATA the address of the variable where the received data block shall be stored is specified via the ADR address operator. The size of the variable must be big enough to store the complete data block (e.g. BYTE array). Furthermore, the format (BYTE, WORD, etc.) of the data must be considered.

DONE BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred during the block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during the request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

ERNO1 BYTE

Output ERNO1 provides an additional DPV1-specific error information in case that an error occurred during processing. ERNO1 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO1 is only valid if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

ERNO1 of the DPV1 blocks is encoded as follows. The upper nibble (the higher significant 4 bits) describes the error class, the lower nibble represents the error cause.

7	6	5	4	3	2	1	0
Error class				Error code			

ERNO1		Error class/Error code
DEC	HEX	
0	0	Reserved
...	...	...
159	9F	Reserved
160	A0	10 Application / 0 Read error
161	A1	10 Application / 1 Write error
162	A2	10 Application / 2 Error module
163	A3	Reserved
...	...	...
167	A7	Reserved
168	A8	10 Application / 8 Version conflict
169	A9	10 Application / 9 Function not supported
170	AA	10 Application / 10 Manufacturer-specific
...	...	...
175	AF	10 Application / 15 Manufacturer-specific
176	B0	11 Access / 0 Invalid index
177	B1	11 Access / 1 Invalid length of data to be written
178	B2	11 Access / 2 Invalid slot
179	B3	11 Access / 3 Type conflict
180	B4	11 Access / 4 Invalid range
181	B5	11 Access / 5 Status conflict
182	B6	11 Access / 6 Access denied
183	B7	11 Access / 7 Invalid value range
184	B8	11 Access / 8 Invalid parameter
185	B9	11 Access / 9 Invalid type
186	BA	11 Access / 10 Manufacturer-specific
...	...	...
191	BF	11 Access / 15 Manufacturer-specific
192	C0	12 Resources / 0 Read conflict
193	C1	12 Resources / 1 Write conflict
194	C2	12 Resources / 2 Resource used
195	C3	12 Resources / 3 Resource not available
196	C4	Reserved
...	...	...
199	C7	Reserved
200	C8	12 Resources / 10 Manufacturer-specific
...	...	...
207	CF	12 Resources / 15 Manufacturer-specific
208	D0	Reserved
...	...	...
255	FF	Reserved

ERNO2 BYTE

Output ERNO2 provides an additional DPV1-specific error information if an error occurred during processing. ERNO2 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO2 is only valid if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC). The encoding of ERNO2 is completely manufacturer-specific.

DATA_LEN BYTE

DATA_LEN outputs the actual length (in bytes) of the data read by the slave. DATA_LEN is only valid if DONE is TRUE and ERR is FALSE. If DATA_LEN contains a value X which is not 0, the block has stored X bytes of data in the variable specified at DATA. For instance, if DATA is a byte array with start index 1, the valid data of the slave are contained in the entries DATA[1] to DATA[X].

Function call in IL

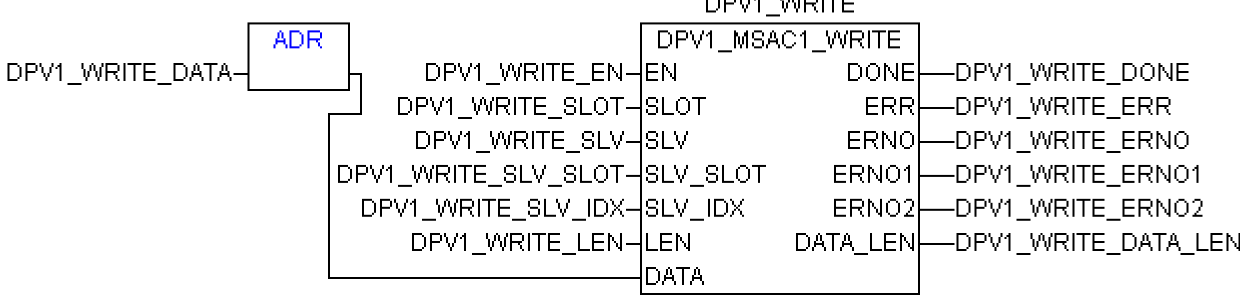
```
LD   DPV1_READ_DATA
ADR
ST   DPV1_READ_DATA_ADR

CAL  DPV1_READ
     (EN   := DPV1_READ_EN,
     SLOT := DPV1_READ_SLOT,
     SLV  := DPV1_READ_SLV,
     SLV_SLOT := DPV1_READ_SLV_SLOT,
     SLV_IDX  := DPV1_READ_SLV_IDX,
     LEN  := DPV1_READ_LEN,
     DATA := DPV1_READ_DATA_ADR)

LD   DPV1_READ.DONE
ST   DPV1_READ_DONE

LD   DPV1_READ.ERR
ST   DPV1_READ_ERR

LD   DPV1_READ.ERNO
ST   DPV1_READ_ERNO

LD   DPV1_READ.ERNO1
ST   DPV1_READ_ERNO1

LD   DPV1_READ.ERNO2
ST   DPV1_READ_ERNO2

LD   DPV1_READ.DATA_LEN
ST   DPV1_READ_DATA_LEN
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
DPV1_READ
     (EN   := DPV1_READ_EN,
     SLOT := DPV1_READ_SLOT,
     SLV  := DPV1_READ_SLV,
     SLV_SLOT := DPV1_READ_SLV_SLOT,
     SLV_IDX  := DPV1_READ_SLV_IDX,
     LEN  := DPV1_READ_LEN,
     DATA := ADR(DPV1_READ_DATA));

DPV1_READ_DONE     := DPV1_READ.DONE;
DPV1_READ_ERR      := DPV1_READ.ERR;
DPV1_READ_ERNO     := DPV1_READ.ERNO;
DPV1_READ_ERNO1    := DPV1_READ.ERNO1;
DPV1_READ_ERNO2    := DPV1_READ.ERNO2;
DPV1_READ_DATA_LEN := DPV1_READ.DATA_LEN;
```

DPV1_MSAC1_WRITE Writing a data block to a DPV1 slave



The block DPV1_MSAC1_WRITE can be used to write a data block to a DPV1 slave by specifying the slot and index.

Block data

Available as of PLC runtime system:	V1.0	Remark:
Included in library:	Profibus_AC500_V10.lib	

Block type

Function block with historical values

Parameter

Instance		DPV1_MSAC1_WRITE	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
SLV	Input	BYTE	DPV1 slave address
SLV_SLOT	Input	BYTE	Slot number of the data block to be written
SLV_IDX	Input	BYTE	Index of the data block to be written
LEN	Input	BYTE	Length of the data block to be written
DATA	Input	DWORD	Memory address for data block (via ADR operator)
DONE	Output	BOOL	Ready message of the block
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
ERNO1	Output	BYTE	Additional error information
ERNO2	Output	BYTE	Additional error information
DATA_LEN	Output	BYTE	Actual length of the written data

Description

The block DPV1_MSAC1_WRITE implements the acyclic PROFIBUS DPV1 service MSAC1_WRITE. Using this function, the master has write access to slot and index-related data of slaves supporting DPV1. DPV1_MSAC1_WRITE works outside the cyclic process data exchange.

Every time a FALSE → TRUE edge is applied to input EN, DPV1_MSAC1_WRITE reads the values at its inputs and the data to be written and sends a corresponding request message to the coupler. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

EN BOOL

If a FALSE → TRUE edge is applied to input EN, all further inputs are read in.

If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at output ERR. Additionally, the termination of the request processing is indicated by DONE = TRUE. While the request is processed, state changes at input EN are recognized but not evaluated.

SLOT BYTE

At input SLOT the coupler slot (module number) is selected which should be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

SLV BYTE

At input SLV the bus address of the DP slave is specified to which the data shall be written. Valid values: 0..126.

SLV_SLOT BYTE

At input SLV_SLOT the number of the slot within the slave is specified to which the data shall be written. Valid values: 0..254.

SLV_IDX BYTE

At input SLV_IDX the number of the index within the slot is specified to which the data shall be written. Valid values: 0..254.

LEN BYTE

At input LEN the length of the data block to be written is specified. Valid values: 0..240.

DATA DWORD

At input DATA the address of the variable containing the data block to be transmitted is specified via the ADR address operator. The size of the variable must be big enough to store the complete data block (e.g. BYTE array). Furthermore, the format (BYTE, WORD, etc.) of the data must be considered.

DONE BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL

Output ERR indicates whether an error occurred during the block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD

Output ERNO provides an error identifier if an invalid value was applied to an input or if an error occurred during the request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. The error messages encoding at output ERNO are explained at the beginning of the library description.

ERNO1 BYTE

Output ERNO1 provides an additional DPV1-specific error information in case that an error occurred during processing. ERNO1 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO1 is only valid if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

ERNO1 of the DPV1 blocks is encoded as follows. The upper nibble (the higher significant 4 bits) describes the error class, the lower nibble represents the error cause.

7	6	5	4	3	2	1	0
Error class				Error code			

ERNO1		Error class/Error code
DEC	HEX	
0	0	Reserved
...	...	...
159	9F	Reserved
160	A0	10 Application / 0 Read error
161	A1	10 Application / 1 Write error
162	A2	10 Application / 2 Error module
163	A3	Reserved
...	...	...
167	A7	Reserved
168	A8	10 Application / 8 Version conflict
169	A9	10 Application / 9 Function not supported
170	AA	10 Application / 10 Manufacturer-specific
...	...	...
175	AF	10 Application / 15 Manufacturer-specific
176	B0	11 Access / 0 Invalid index
177	B1	11 Access / 1 Invalid length of data to be written
178	B2	11 Access / 2 Invalid slot
179	B3	11 Access / 3 Type conflict
180	B4	11 Access / 4 Invalid range
181	B5	11 Access / 5 Status conflict
182	B6	11 Access / 6 Access denied
183	B7	11 Access / 7 Invalid value range
184	B8	11 Access / 8 Invalid parameter
185	B9	11 Access / 9 Invalid type
186	BA	11 Access / 10 Manufacturer-specific
...	...	...
191	BF	11 Access / 15 Manufacturer-specific
192	C0	12 Resources / 0 Read conflict
193	C1	12 Resources / 1 Write conflict
194	C2	12 Resources / 2 Resource used
195	C3	12 Resources / 3 Resource not available
196	C4	Reserved
...	...	...
199	C7	Reserved
200	C8	12 Resources / 10 Manufacturer-specific
...	...	...
207	CF	12 Resources / 15 Manufacturer-specific
208	D0	Reserved
...	...	...
255	FF	Reserved

ERNO2 BYTE

Output ERNO2 provides an additional DPV1-specific error information if an error occurred during processing. ERNO2 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO2 is only valid if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC). The encoding of ERNO2 is completely manufacturer-specific.

DATA_LEN BYTE

The output DATA_LEN displays the actual length (in bytes) of the data written to the slave. DATA_LEN is only valid if DONE = TRUE and ERR = 0.

Function call in IL

```
LD   DPV1_WRITE_DATA
ADR
ST   DPV1_WRITE_DATA_ADR

CAL  DPV1_WRITE
     (EN   := DPV1_WRITE_EN,
     SLOT := DPV1_WRITE_SLOT,
     SLV  := DPV1_WRITE_SLV,
     SLV_SLOT := DPV1_WRITE_SLV_SLOT,
     SLV_IDX  := DPV1_WRITE_SLV_IDX,
     LEN  := DPV1_WRITE_LEN,
     DATA := DPV1_WRITE_DATA_ADR)

LD   DPV1_WRITE.DONE
ST   DPV1_WRITE_DONE

LD   DPV1_WRITE.ERR
ST   DPV1_WRITE_ERR

LD   DPV1_WRITE.ERNO
ST   DPV1_WRITE_ERNO

LD   DPV1_WRITE.ERNO1
ST   DPV1_WRITE_ERNO1

LD   DPV1_WRITE.ERNO2
ST   DPV1_WRITE_ERNO2

LD   DPV1_WRITE.DATA_LEN
ST   DPV1_WRITE_DATA_LEN
```

Note:
In IL, the function call has to be written in one line.

Function call in ST

```
DPV1_WRITE
     (EN   := DPV1_WRITE_EN,
     SLOT := DPV1_WRITE_SLOT,
     SLV  := DPV1_WRITE_SLV,
     SLV_SLOT := DPV1_WRITE_SLV_SLOT,
     SLV_IDX  := DPV1_WRITE_SLV_IDX,
     LEN  := DPV1_WRITE_LEN,
     DATA := ADR(DPV1_WRITE_DATA));

DPV1_WRITE_DONE     := DPV1_WRITE.DONE;
DPV1_WRITE_ERR      := DPV1_WRITE.ERR;
DPV1_WRITE_ERNO     := DPV1_WRITE.ERNO;
DPV1_WRITE_ERNO1    := DPV1_WRITE.ERNO1;
DPV1_WRITE_ERNO2    := DPV1_WRITE.ERNO2;
DPV1_WRITE_DATA_LEN := DPV1_WRITE.DATA_LEN;
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

Components of the PROFIBUS DP Library 2

D

DPM_CTRL Sending global control commands to the DP slave 5

DPM_READ_INPUT Reading the input data of a slave which is not assigned to the master 11

DPM_READ_OUTPUT Reading the output data of a slave which is not assigned to the master 14

DPM_SET_PRM Sending user parameters to a DP slave 17

DPM_SLV_DIAG Polling detailed diagnostic data of a DP slave 20

DPM_STAT Reading out the status of the PROFIBUS coupler 30

DPM_SYS_DIAG Reading out a status overview of all DP slaves 37

DPV1_MSAC1_READ Reading a data block from a DPV1 slave 40

DPV1_MSAC1_WRITE Writing a data block to a DPV1 slave 45

G

Glossary 50

O

Overview of blocks arranged according to their call names 4

P

Precondition for the use of the PROFIBUS DP Library 2

S

STATIONSTATUS_1_TYPE 21, 24

STATIONSTATUS_2_TYPE 22, 25

STATIONSTATUS_3_TYPE 22, 26

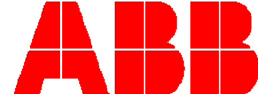
Software Description

AC500

Scalable PLC
for Individual Automation

CANopen
Function Block Library

CANopen



Contents

CANopen Library 2

Special characteristics of the CANopen Library 2

Components of the CANopen Library 2

General notes regarding CAN blocks 3

Overview of blocks arranged according to their call names 5

CAN2A_INFO Reading information about CAN 2.0A communication 6

CAN2A_REC Reading CAN 2.0A telegrams (with 11 bit identifier) from a receive buffer 9

CAN2A_SEND Transmitting CAN 2.0A telegrams (with 11 bit identifier) 13

CAN2B_INFO Reading information about CAN 2.0B communication 16

CAN2B_REC Reading CAN 2.0B telegrams (with 29 bit identifier) from a receive buffer 19

CAN2B_SEND Transmitting CAN 2.0B telegrams (with 29 bit identifier) 23

CANOM_NMT Controlling NMT node states via network management 26

CANOM_NODE_DIAG Polling diagnosis data from a slave 29

CANOM_RES_ERR Resetting the coupler's error indications 36

CANOM_SDO_READ Reading the value of a slave object 39

CANOM_SDO_WRITE Writing the value of a slave object 43

CANOM_STATE Reading the CANopen coupler status 47

CANOM_SYS_DIAG Displaying status surveys of all slaves 54

Glossary 57

Index 59

CANopen Library

Special characteristics of the CANopen library

Note: CANopen communication is only performed in RUN mode of the PLC, but not in simulation mode.

The function blocks contained in the CANopen library access both, the PLC run time system as well as directly the coupler. The definitions and functions required for this are stored in the internal library SysExt_AC500_V10.LIB. This library is automatically included during project setup.

The library contains function blocks for a comfortable handling of the CANopen coupler as well as definitions of various data types. These structures enable a clear presentation of data sets.

Components of the CANopen library

Function blocks

The CANopen library contains the following function blocks:

Group: CAN 2.0A	
CAN2A_INFO	Reading information for CAN 2.0A communication
CAN2A_REC	Reading CAN 2.0A telegrams (with 11 bit identifier) from a receive buffer
CAN2A_SEND	Transmitting CAN 2.0A telegrams (with 11 bit identifier)

Group: CAN 2.0B	
CAN2B_INFO	Reading information for CAN 2.0B communication
CAN2B_REC	Reading CAN 2.0B telegrams (with 29 bit identifier) from a receive buffer
CAN2B_SEND	Transmitting CAN 2.0B telegrams (with 29 bit identifier)

Group: Control	
CANOM_NMT	Controlling NMT node states via network management

Group: Diagnosis	
CANOM_NODE_DIAG	Polling diagnosis data from a slave
CANOM_RES_ERR	Resetting the coupler's error indications
CANOM_STATE	Reading the CANopen coupler status
CANOM_SYS_DIAG	Displaying status information of all slaves

Group: Parameters	
CANOM_SDO_READ	Reading the value of a slave object
CANOM_SDO_WRITE	Writing the value of a slave object

Data types

The following data types (structures) are defined in the CANopen library:

Group: CAN	
CAN2A_MESSAGE_TYPE	Telegram structure according to CAN 2.0A
CAN2B_MESSAGE_TYPE	Telegram structure according to CAN 2.0B

Group: CANopen	
CANOM_COM_ERR_TYPE	Communication error
CANOM_EMCY_TYPE	Emergency telegram
CANOM_NODESTATUS_1_TYPE	Node diagnosis
CANOM_STATE_BITS_TYPE	Bits for coupler state description

The data type descriptions for the CANopen master can be found at the corresponding block descriptions. The CAN data types are described in section "General notes regarding CAN blocks".

General notes regarding CAN blocks

The CAN blocks represent additional functionality. They are not required for normal CANopen operation.

Data exchange between the controller working as a CANopen master and the connected nodes is normally configured using SYCON.net. The configured nodes are taken into operation by the coupler when the program starts. During running operation, data exchange is performed automatically. The I/O data of the nodes can be accessed like variables without any use of additional blocks. If the PLC program is stopped, the master shuts down the bus in a controlled manner. The CAN blocks can be used to implement further functions which are not contained in the configuration.

The CAN blocks are able to transmit and receive any CAN telegrams. Messages with 11 bit identifiers according to CAN 2.0 A are as well supported as messages with 29 bit identifiers according to CAN 2.0 B.

CAN is the basis for various other protocols (SDS, DeviceNet, CANopen). A common point of all protocols is the telegram structure consisting of 11 bit or 29 bit identifiers and up to 8 data bytes. Use and meaning of the individual telegram components differ from protocol to protocol. Therefore, the transmitted and received telegrams are forwarded transparently and not interpreted by the blocks to provide universal use of the CAN blocks. For a more comfortable handling of CANopen telegrams, additional auxiliary functions and data structures are available.

CAN2A_MESSAGE_TYPE

The library contains one general data type CAN2A_MESSAGE_TYPE to describe a CAN 2.0 A telegram with an 11 bit identifier. This data type is declared as follows:

```
TYPE CAN2A_MESSAGE_TYPE:
STRUCT
    ID: WORD;
    RTR: BOOL;
    DLC: BYTE;
    DATA: ARRAY [1..8] OF BYTE;
END_STRUCT
END_TYPE
```

ID WORD (identifier)

ID contains the general 11 bit identifier, its value range reaches from 0 to 2047 (16#0 to 16#7FF).

RTR BOOL (remote transmission request)

RTR contains the RTR bit in the telegram header.

DLC BYTE (data length code)

DLC contains the data length code in the telegram header and specifies the valid length in bytes for the user data following in DATA. Valid values for DLC are 0 to 8.

DATA ARRAY[1..8] OF BYTE (data)

DATA contains the telegram data (if available). A CAN telegram can contain 0 to 8 bytes of data. The actual data length of a telegram is described by the data length code (DLC) contained in the telegram header. In DATA, only the first bytes are valid as specified by the DLC.

CAN2B_MESSAGE_TYPE

The library contains one general data type CAN2B_MESSAGE_TYPE to describe a CAN 2.0 B telegram with a 29 bit identifier. This data type is declared as follows:

```
TYPE CAN2B_MESSAGE_TYPE:
STRUCT
    ID: DWORD;
    RTR: BOOL;
    DLC: BYTE;
    DATA: ARRAY [1..8] OF BYTE;
END_STRUCT
END_TYPE
```

ID DWORD (identifier)

ID contains the general 29 bit identifier, its value range reaches from
0 to 536870911 (16#0 to 16# 1FFFFFFF).

RTR BOOL (remote transmission request)

RTR contains the RTR bit in the telegram header.

DLC BYTE (data length code)

DLC contains the data length code in the telegram header and specifies the valid length in bytes for the user data following in DATA. Valid values for DLC are 0 to 8.

DATA ARRAY[1..8] OF BYTE (data)

DATA contains the telegram data (if available). A CAN telegram can contain 0 to 8 bytes of data. The actual data length of a telegram is described by the data length code (DLC) contained in the telegram header. In DATA, only the first bytes are valid as specified by the DLC.

Overview of blocks arranged according to their call names

Character description:

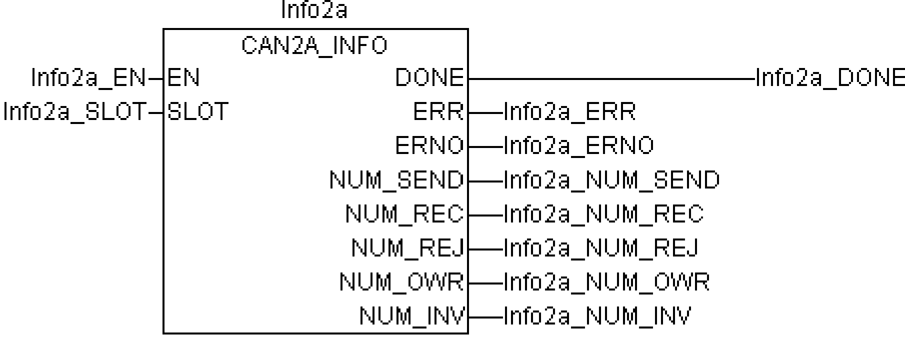
FBhv … Function block with historical values

FBnohv … Function block without historical values

F … Function

VE name	Type	Function
CAN2A_INFO	FBhv	Reading information about CAN 2.0A communication
CAN2A_REC	FBhv	Reading CAN 2.0A telegrams (with 11 bit identifier) from a receive buffer
CAN2A_SEND	FBhv	Transmitting CAN 2.0A telegrams (with 11 bit identifier)
CAN2B_INFO	FBhv	Reading information about CAN 2.0B communication
CAN2B_REC	FBhv	Reading CAN 2.0B telegrams (with 29 bit identifier) from a receive buffer
CAN2B_SEND	FBhv	Transmitting CAN 2.0B telegrams (with 29 bit identifier)
CANOM_NMT	FBhv	Controlling NMT node states via network management
CANOM_NODE_DIAG	FBhv	Polling diagnosis data from a slave
CANOM_RES_ERR	FBhv	Resetting the coupler's error indications
CANOM_SDO_READ	FBhv	Reading the value of a slave object
CANOM_SDO_WRITE	FBhv	Writing the value of a slave object
CANOM_STATE	FBhv	Reading the CANopen coupler status
CANOM_SYS_DIAG	FBhv	Displaying status information of all slaves

CAN2A_INFO Reading information about CAN 2.0A communication



The block CAN2A_INFO outputs information concerning the status of the CAN 2.0A communication.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CAN2A_INFO	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NUM_SEND	Output	DWORD	Number of sent CAN 2.0A telegrams
NUM_REC	Output	DWORD	Number of received CAN 2.0A telegrams
NUM_REJ	Output	DWORD	Number of rejected CAN 2.0A telegrams
NUM_OWR	Output	DWORD	Number of overwritten CAN 2.0A telegrams
NUM_INV	Output	DWORD	Number of faulty CAN 2.0A telegrams

Description

Using the block CAN2A_INFO, various status information about the CAN 2.0A communication can be read.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs. If the block detects a CAN coupler with a configured CAN 2.0A protocol at the specified SLOT, this is indicated by DONE = TRUE and ERR = FALSE. The corresponding status information is then available at the block outputs.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected to used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

NUM_SEND DWORD (number of sent telegrams)

Output NUM_SEND displays the number of transmitted CAN 2.0A telegrams. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REC DWORD (number of received telegrams)

Output NUM_REC displays the number of received CAN 2.0A telegrams, regardless of whether a corresponding buffer has been set via the PLC configuration or not. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REJ DWORD (number of rejected telegrams)

Output NUM_REJ displays the number of CAN 2.0A telegrams rejected due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_OWR DWORD (number of overwritten telegrams)

Output NUM_OWR displays the number of CAN 2.0A telegrams overwritten by a new incoming telegram due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_INV DWORD (number of invalid/unknown telegrams)

Output NUM_INV displays the number of received CAN 2.0A telegrams that could not be assigned to any buffer defined in the controller configuration. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

Function call in IL

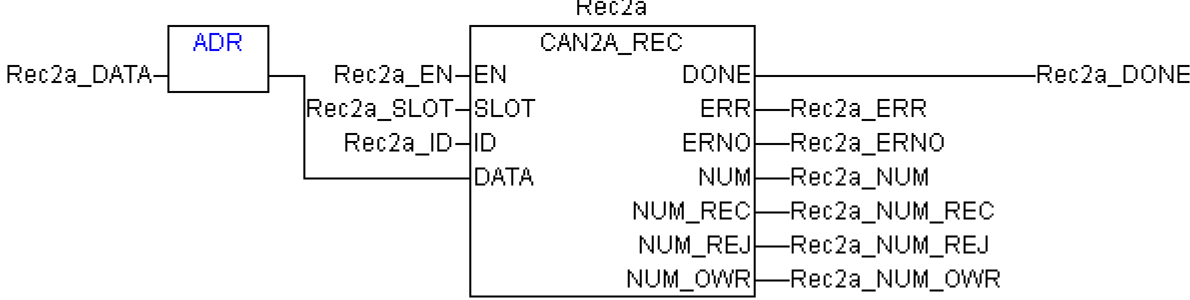
```
CAL Info2a (
    EN   := Info2a_EN,
    SLOT := Info2a_SLOT)

LD  Info2a.DONE
ST  Info2a_DONE

LD  Info2a.ERR
ST  Info2a_ERR

LD  Info2a.ERNO
ST  Info2a_ERNO

LD  Info2a.NUM_SEND
ST  Info2a_NUM_SEND

LD  Info2a.NUM_REC
ST  Info2a_NUM_REC

LD  Info2a.NUM_REJ
ST  Info2a_NUM_REJ

LD  Info2a.NUM_OWR
ST  Info2a_NUM_OWR

LD  Info2a.NUM_INV
ST  Info2a_NUM_INV
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
Info2a ( EN     := Info2a_EN,
         SLOT   := Info2a_SLOT);

Info2a_DONE     := Info2a.DONE;
Info2a_ERR      := Info2a.ERR;
Info2a_ERNO     := Info2a.ERNO;
Info2a_NUM_SEND := Info2a.NUM_SEND;
Info2a_NUM_REC  := Info2a.NUM_REC;
Info2a_NUM_REJ  := Info2a.NUM_REJ;
Info2a_NUM_OWR  := Info2a.NUM_OWR;
Info2a_NUM_INV  := Info2a.NUM_INV;
```

CAN2A_REC Reading CAN 2.0A telegrams (with 11 bit identifier) from a receive buffer



Using CAN2A_REC, any CAN telegrams with 11 bit identifiers according to CAN 2.0A can be read from a receive buffer.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block block with historical values

Parameters

Instance		CAN2A_REC	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
ID	Input	WORD	Identifier of the buffer to be read
DATA	Input	DWORD	Address from which on the received CAN 2.0A telegrams are stored (via ADR operator)
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NUM	Output	WORD	Number of CAN 2.0A telegrams contained in the buffer of the selected identifier
NUM_REC	Output	DWORD	Number of received CAN 2.0A telegrams
NUM_REJ	Output	DWORD	Number of rejected CAN 2.0A telegrams
NUM_OWR	Output	DWORD	Number of overwritten CAN 2.0A telegrams

Description

Using CAN2A_REC, CAN telegrams with 11 bit identifiers according to CAN 2.0A can be received. Only those telegrams can be received, the identifiers of which have been enabled previously. Enabling the individual identifiers is done in the controller configuration. For each identifier a corresponding buffer, the buffer size and its behavior if the buffer is full has to be defined. Received telegrams with non-enabled identifiers are automatically rejected by the coupler (see CAN2A_INFO, output NUM_INV).

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active and the values at its block inputs are valid, the CAN2A telegrams temporarily stored in the buffer are read one after another. After a telegram has been read without errors, DONE is set to TRUE and ERR is set to FALSE for one cycle.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) is selected to be used by the block.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

ID WORD (identifier)

Input ID is used to specify the identifier of the CAN 2.0A telegrams to be read from the buffer. If no buffer has been specified for the selected identifier using the controller configuration, this is indicated accordingly at the block outputs.

DATA DWORD (data)

Input DATA is used to specify the address starting from which the received CAN 2.0A telegrams should be written. Usually, this specification is done via the ADR operator and should point to variables of the type CAN2A_MESSAGE_TYPE or CANOpen_MESSAGE_TYPE.

DONE BOOL (done)

Output DONE indicates the processing state of the block. If a telegram has been read or if the read process has been aborted due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

NUM WORD (number of telegrams)

Output NUM displays the number of CAN 2.0A telegrams not yet read from the buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value.

NUM_REC DWORD (number of received telegrams)

Output NUM_REC displays the total number of CAN 2.0A telegrams received in the buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value.

NUM_REJ DWORD (number of rejected telegrams)

Output NUM_REJ displays the total number of received CAN 2.0A telegrams rejected due to a full buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value. Whether incoming telegrams are generally rejected in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_OWR DWORD (number of overwritten telegrams)

Output NUM_OWR displays the total number of CAN 2.0A telegrams overwritten by a new incoming telegram due to the full buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value. Whether incoming telegrams are generally rejected in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

Function call in IL

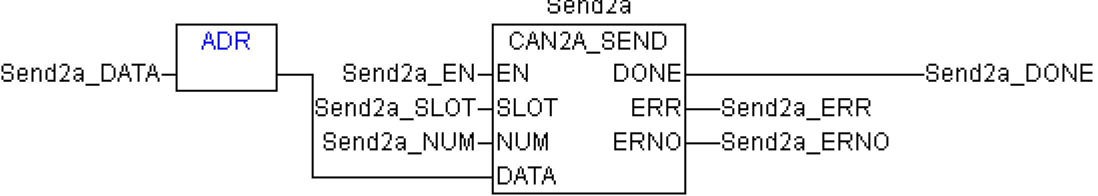
```
LD  Rec2a_DATA
ADR
ST  ADR_Rec2a_DATA

CAL Rec2a (
    EN   := Rec2a_EN,
    SLOT := Rec2a_SLOT
    ID   := Rec2a_ID
    DATA := ADR_Rec2a_DATA)

LD  Rec2a.DONE
ST  Rec2a_DONE

LD  Rec2a.ERR
ST  Rec2a_ERR

LD  Rec2a.ERNO
ST  Rec2a_ERNO

LD  Rec2a.NUM
ST  Rec2a_NUM

LD  Rec2a.NUM_REC
ST  Rec2a_NUM_REC

LD  Rec2a.NUM_REJ
ST  Rec2a_NUM_REJ

LD  Rec2a.NUM_OWR
ST  Rec2a_NUM_OWR
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
Rec2a (   EN   := Rec2a_EN,
          SLOT := Rec2a_SLOT,
          ID   := Rec2a_ID,
          DATA := ADR(Rec2a_DATA));

Rec2a_DONE      := Rec2a.DONE;
Rec2a_ERR       := Rec2a.ERR;
Rec2a_ERNO      := Rec2a.ERNO;
Rec2a_NUM       := Rec2a.NUM;
Rec2a_NUM_REC   := Rec2a.NUM_REC;
Rec2a_NUM_REJ   := Rec2a.NUM_REJ;
Rec2a_NUM_OWR   := Rec2a.NUM_OWR;
```

CAN2A_SEND Transmitting CAN 2.0A telegrams (with 11 bit identifier)



Using CAN2A_SEND, any CAN telegrams with 11 bit identifiers according to CAN 2.0A can be transmitted.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CAN2A_SEND	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
NUM	Input	BYTE	Number of CAN 2.0A telegrams to be transmitted
DATA	Input	DWORD	Address from which on the CAN 2.0A telegrams to be transmitted are stored (via ADR operator)
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

Using CAN2A_SEND, CAN telegrams with 11 bit identifiers according to CAN 2.0A can be transmitted.

Every time a FALSE->TRUE edge is applied to input EN, CAN2A_SEND reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE and ERR = FALSE. A possibly occurring error is indicated by output ERR = TRUE.

The block is able to transmit several telegrams up to a total length of 254 bytes within one event to the coupler which in turn transmits these telegrams sequentially via the bus. Thus, the maximum number of simultaneously transmitted telegrams depends on the sum of the individual telegram lengths. If all telegrams to be transmitted do not contain any other data than the 2 header bytes (identifier, RTR and DLC; Data Length Code DLC = 0), up to 127 telegrams can be transmitted at the same time (2 x 127 = 254). However, if all telegrams contain the maximum 8 bytes of data, only up to 25 telegrams can be transmitted to the coupler simultaneously ((2 + 8) x 25 = 250).

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at output ERR and the termination of the request processing is indicated by DONE = TRUE.

During processing of the request, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

NUM BYTE (number)

Input NUM is used to specify the number of valid telegrams to be transmitted and stored starting at address DATA. Basically, the valid values for NUM are 1 to 127. However, the upper limit depends on the total length of all telegrams. The total length is calculated by the block from the data length codes (DLC) of the individual telegrams and must not exceed 254 bytes. Otherwise an error message is generated. In such cases, the number of telegrams to be transmitted has to be chosen correspondingly that the total length does not exceed 254.

DATA DWORD (data)

At input DATA the address of the variable containing the telegrams to be transmitted is specified. This specification is usually done via the ADR operator.

The telegrams to be transmitted have to be of the data type CAN2A_MESSAGE_TYPE or CANopen_MESSAGE_TYPE defined in the library. If only one telegram has to be transmitted (NUM = 1), a single variable is sufficient. If more than one telegram have to be transmitted, the chosen variable has to be of the type ARRAY [1..X] CAN2A_MESSAGE_TYPE or ARRAY [1..X] OF CANopen_MESSAGE_TYPE with the value of X at least as high as the number specified at input NUM. Then, the first element of the ARRAY or directly the ARRAY (without index) has to be specified at the input of the ADR operator.

The block only evaluates the data length codes (DLC) of the individual telegrams to determine the total length over all telegrams and does not perform any further interpretations of the telegrams. Since the position within the telegram header as well as the encoding of the data length code are identical for all CAN telegrams, the telegrams to be transmitted can be alternatively specified in the general format (CAN2A_MESSAGE_TYPE) or in the CANopen format (CANopen_MESSAGE_TYPE).

DONE BOOL (done)

Output DONE indicates the processing state of the block. If telegram transmission is completed or if transmission has been aborted due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

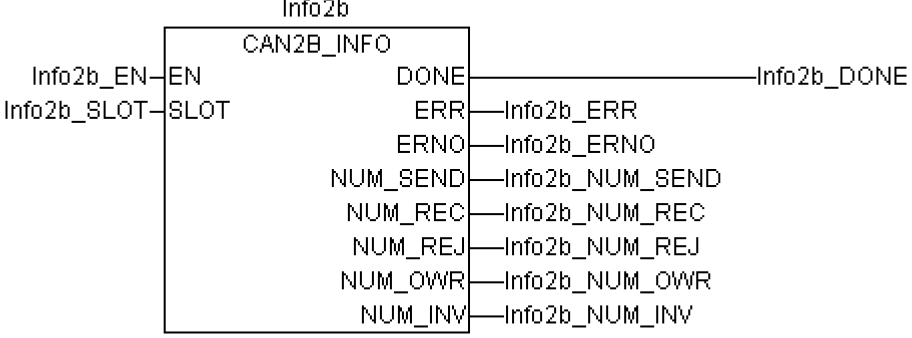
```
LD   Send2a_DATA
ADR
ST   ADR_Send2a_DATA

CAL Send2a (
    EN   := Send2a_EN,
    SLOT := Send2a_SLOT
    NUM  := Send2a_NUM
    DATA := ADR_Send2a_DATA)

LD   Send2a.DONE
ST   Send2a_DONE

LD   Send2a.ERR
ST   Send2a_ERR

LD   Send2a.ERNO
ST   Send2a_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
Send2a (  EN   := Send2a_EN,
          SLOT := Send2a_SLOT,
          NUM  := Send2a_NUM,
          DATA := ADR(Send2a_DATA));

Send2a_DONE     := Send2a.DONE;
Send2a_ERR      := Send2a.ERR;
Send2a_ERNO     := Send2a.ERNO;
```

CAN2B_INFO Reading information about CAN 2.0B communication



The block CAN2B_INFO outputs information concerning the status of the CAN 2.0B communication.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CAN2B_INFO	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NUM_SEND	Output	DWORD	Number of sent CAN 2.0B telegrams
NUM_REC	Output	DWORD	Number of received CAN 2.0B telegrams
NUM_REJ	Output	DWORD	Number of rejected CAN 2.0B telegrams
NUM_OWR	Output	DWORD	Number of overwritten CAN 2.0B telegrams
NUM_INV	Output	DWORD	Number of faulty CAN 2.0B telegrams

Description

Using CAN2B_INFO, various status information about the CAN 2.0B communication can be read.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs. If the block detects a CAN coupler with a configured CAN 2.0B protocol at the specified SLOT, this is indicated by DONE = TRUE and ERR = FALSE. The corresponding status information are then available at the block outputs.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing due to an error, DONE is set to TRUE. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

NUM_SEND DWORD (number of sent telegrams)

Output NUM_SEND displays the number of transmitted CAN 2.0B telegrams. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REC DWORD (number of received telegrams)

Output NUM_REC displays the number of received CAN 2.0B telegrams, regardless of whether a corresponding buffer has been set via the PLC configuration or not. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REJ DWORD (number of rejected telegrams)

Output NUM_REJ displays the number of CAN 2.0B telegrams rejected due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally rejected in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_OWR DWORD (number of overwritten telegrams)

Output NUM_OWR displays the number of CAN 2.0B telegrams overwritten by a new incoming telegram due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally rejected in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_INV DWORD (number of invalid/unknown telegrams)

Output NUM_INV displays the number of received CAN 2.0B telegrams that could not be assigned to any buffer defined in the controller configuration. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

Function call in IL

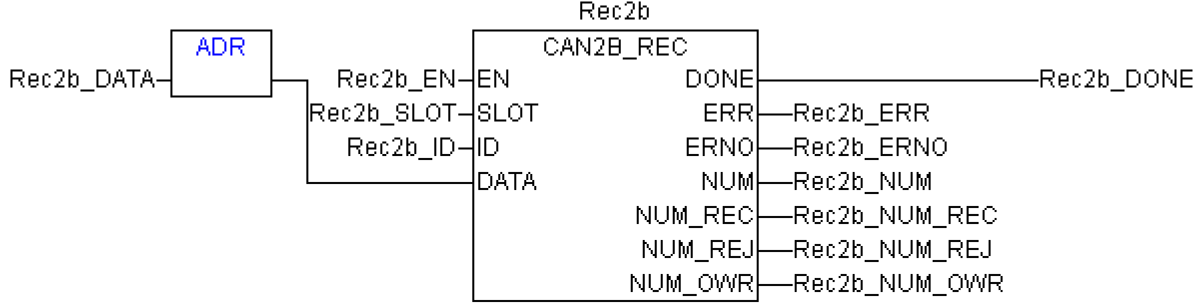
```
CAL Info2b (
    EN   := Info2b_EN,
    SLOT := Info2b_SLOT)

LD  Info2b.DONE
ST  Info2b_DONE

LD  Info2b.ERR
ST  Info2b_ERR

LD  Info2b.ERNO
ST  Info2b_ERNO

LD  Info2b.NUM_SEND
ST  Info2b_NUM_SEND

LD  Info2b.NUM_REC
ST  Info2b_NUM_REC

LD  Info2b.NUM_REJ
ST  Info2b_NUM_REJ

LD  Info2b.NUM_OWR
ST  Info2b_NUM_OWR

LD  Info2b.NUM_INV
ST  Info2b_NUM_INV
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
Info2b ( EN      := Info2b_EN,
         SLOT    := Info2b_SLOT);

Info2b_DONE     := Info2b.DONE;
Info2b_ERR      := Info2b.ERR;
Info2b_ERNO     := Info2b.ERNO;
Info2b_NUM_SEND := Info2b.NUM_SEND;
Info2b_NUM_REC  := Info2b.NUM_REC;
Info2b_NUM_REJ  := Info2b.NUM_REJ;
Info2b_NUM_OWR  := Info2b.NUM_OWR;
Info2b_NUM_INV  := Info2b.NUM_INV;
```

CAN2B_REC Reading CAN 2.0B telegrams (with 29 bit identifier) from a receive buffer



Using CAN2B_REC, any CAN telegrams with 29 bit identifiers according to CAN 2.0B can be read from a receive buffer.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CAN2B_REC	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
ID	Input	WORD	Identifier of the buffer to be read.
DATA	Input	DWORD	Address from which on the received CAN 2.0B telegrams are stored (via ADR operator)
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NUM	Output	WORD	Number of CAN 2.0B telegrams contained in the buffer of the selected identifier
NUM_REC	Output	DWORD	Number of received CAN 2.0B telegrams
NUM_REJ	Output	DWORD	Number of rejected CAN 2.0B telegrams
NUM_OWR	Output	DWORD	Number of overwritten CAN 2.0B telegrams

Description

Using CAN2B_REC, CAN telegrams with 29 bit identifiers according to CAN 2.0 B can be received. Only those telegrams can be received, the identifiers of which have been enabled previously. Enabling of this block is done using the configuration tool SYCON.net or using the controller configuration of the CPU.

First, the exchange of CAN 2.0 B telegrams is generally enabled in the coupler using SYCON.net. By means of an acceptance code and an acceptance mask it is furthermore possible to set up an ID filter. The coupler then compares all 29 bit identifiers on the bus with these two entries and forwards the filtered telegrams to the CPU.

Enabling of the individual identifiers is done by the controller configuration of the CPU. For each identifier a corresponding buffer, the buffer size and its behavior if the buffer is full has to be defined. Received telegrams with non-enabled identifiers are automatically dismissed (see CAN2B_INFO, output NUM_INV).

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active and the values at the block inputs are valid, the CAN2B telegrams temporarily stored in the buffer are read one after another. After a telegram has been read without errors, DONE is set to TRUE and ERR is set to FALSE for one cycle.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

ID WORD (identifier)

Input ID is used to specify the identifier of the CAN 2.0B telegrams to be read from the buffer. If no buffer has been specified for the selected identifier using the controller configuration, this is indicated accordingly at the block outputs.

DATA DWORD (data)

Input DATA is used to specify the address starting from which the received CAN 2.0B telegrams should be written. Specification is usually done by means of the ADR operator and should point to variables of the type CAN2B_MESSAGE_TYPE.

DONE BOOL (done)

Output DONE indicates the processing state of the block. If a telegram has been read or if the read process has been aborted due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

NUM WORD (<u>num</u>ber of telegrams)

Output NUM displays the number of CAN 2.0B telegrams not yet read from the buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value.

NUM_REC DWORD (<u>num</u>ber of <u>rec</u>eived telegrams)

Output NUM_REC displays the total number of CAN 2.0B telegrams received in the buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value.

NUM_REJ DWORD (<u>num</u>ber of <u>rej</u>ected telegrams)

Output NUM_REJ displays the total number of received CAN 2.0B telegrams rejected due to a full buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value. Whether incoming telegrams are generally rejected in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_OWR DWORD (<u>num</u>ber of <u>o</u>ver<u>wr</u>itten telegrams)

Output NUM_OWR displays the total number of CAN 2.0B telegrams overwritten by a new incoming telegram due to the full buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value. Whether incoming telegrams are generally rejected in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

Function call in IL

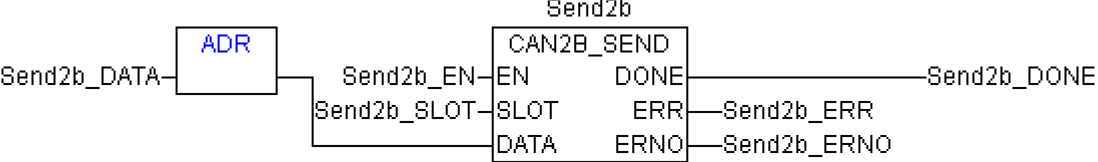
```
LD   Rec2b_DATA
ADR
ST   ADR_Rec2b_DATA

CAL Rec2b (
    EN   := Rec2b_EN,
    SLOT := Rec2b_SLOT
    ID   := Rec2b_ID
    DATA := ADR_Rec2b_DATA)

LD   Rec2b.DONE
ST   Rec2b_DONE

LD   Rec2b.ERR
ST   Rec2b_ERR

LD   Rec2b.ERNO
ST   Rec2b_ERNO

LD   Rec2b.NUM
ST   Rec2b_NUM

LD   Rec2b.NUM_REC
ST   Rec2b_NUM_REC

LD   Rec2b.NUM_REJ
ST   Rec2b_NUM_REJ

LD   Rec2b.NUM_OWR
ST   Rec2b_NUM_OWR
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
Rec2b (   EN   := Rec2b_EN,
          SLOT := Rec2b_SLOT,
          ID   := Rec2b_ID,
          DATA := ADR(Rec2b_DATA));

Rec2b_DONE      := Rec2b.DONE;
Rec2b_ERR       := Rec2b.ERR;
Rec2b_ERNO      := Rec2b.ERNO;
Rec2b_NUM       := Rec2b.NUM;
Rec2b_NUM_REC   := Rec2b.NUM_REC;
Rec2b_NUM_REJ   := Rec2b.NUM_REJ;
Rec2b_NUM_OWR   := Rec2b.NUM_OWR;
```

CAN2B_SEND Transmitting CAN 2.0B telegrams (with 29 bit identifier)



Using CAN2B_SEND, any CAN telegrams with 29 bit identifiers according to CAN 2.0B can be transmitted.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CAN2B_SEND	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DATA	Input	DWORD	Address from which on the CAN 2.0B telegrams to be transmitted are stored (via ADR operator)
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

Using CAN2B_SEND, CAN telegrams with 29 bit identifiers according to CAN 2.0B can be transmitted.

Every time a FALSE->TRUE edge is applied to input EN, CAN2B_SEND reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE and ERR = FALSE. A possibly occurring error is indicated by output ERR = TRUE.

Only one CAN 2.0B can be sent with each transmission.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at output ERR and the termination of the request processing is indicated by DONE = TRUE.

During processing of the request, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DATA DWORD (data)

The telegram to be transmitted or the variable have to be of the data type CAN2B_MESSAGE_TYPE as defined in the library.

DONE BOOL (done)

Output DONE indicates the processing state of the block. If telegram transmission is completed or if transmission has been aborted due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

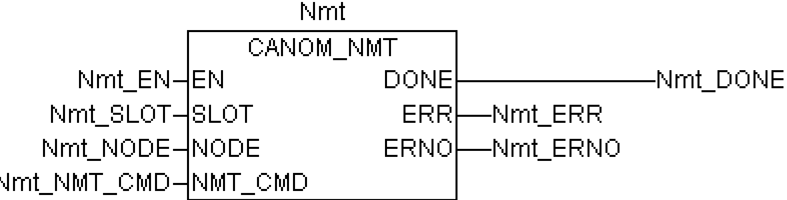
```
LD   Send2b_DATA
ADR
ST   ADR_Send2b_DATA

CAL Send2b (
    EN   := Send2b_EN,
    SLOT := Send2b_SLOT
    DATA := ADR_Send2b_DATA)

LD   Send2b.DONE
ST   Send2b_DONE

LD   Send2b.ERR
ST   Send2b_ERR

LD   Send2b.ERNO
ST   Send2b_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
Send2b (  EN   := Send2b_EN,
          SLOT := Send2b_SLOT,
          DATA := ADR(Send2b_DATA));

Send2b_DONE    := Send2b.DONE;
Send2b_ERR     := Send2b.ERR;
Send2b_ERNO    := Send2b.ERNO;
```

CANOM_NMT Controlling NMT node states via network management



CANOM_NMT can be used to control the operating condition(s) of one specific or all slaves.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CANOM_NMT	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
NODE	Input	BYTE	Node address of the concerning slave
NMT_CMD	Input	BYTE	NMT command according to CAL profile
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

CANOM_NMT can be used to control the operating condition(s) of one specific or all slaves.

Every time a FALSE->TRUE edge is applied to input EN, CANOM_NMT reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

Normally, control of the slave operating states is performed by the automatic control of the CANopen coupler used as NMT master. However, for special applications it can be required to change the state of a specific slave 'manually'. This functionality can be achieved using CANOM_NMT.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at the outputs ERR/ERNO and the termination of the request processing is indicated by DONE = TRUE.

During the processing of a request, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

NODE BYTE (node)

At input NODE the node address of the slave is specified the operating condition of which shall be changed. By applying a value of 1 to 127 to input NODE, one specific slave can be called with the corresponding node address. If NODE = 0, the command is sent to all slaves.

NMT_CMD BYTE (NMT command)

At input NMT_CMD, the NMT command to be sent is specified. The following NMT commands are defined:

NMT command	Meaning
1	Start remote node (slave)
2	Stop remote node (slave)
128	Enter pre-operational: Set slave to pre-operational mode
129	Reset node (slave)
130	Reset communication

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

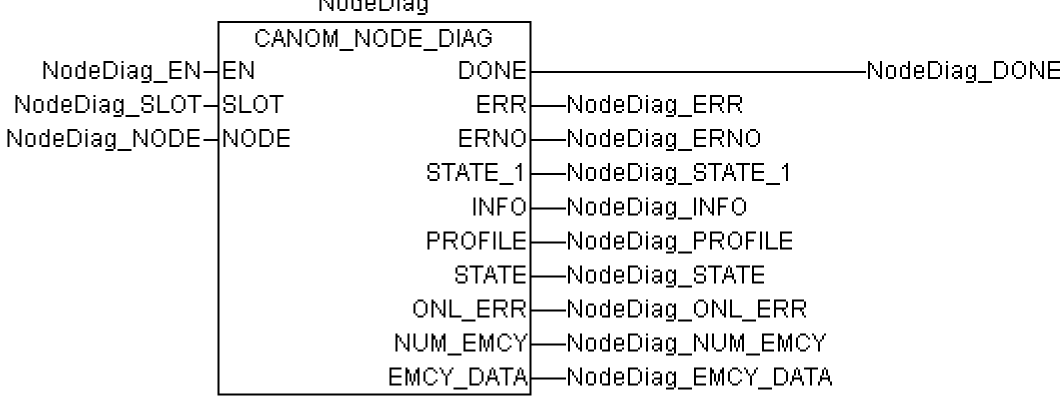
```
CAL Nmt (
    EN      := Nmt_EN,
    SLOT    := Nmt_SLOT
    NODE    := Nmt_NODE
    NMT_CMD := Nmt_NMT_CMD)

LD  Nmt.DONE
ST  Nmt_DONE

LD  Nmt.ERR
ST  Nmt_ERR

LD  Nmt.ERNO
ST  Nmt_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
Nmt( EN      := Nmt_EN,
     SLOT    := Nmt_SLOT,
     NODE    := Nmt_NODE,
     NMT_CMD := Nmt_NMT_CMD);

Nmt_DONE     := Nmt.DONE;
Nmt_ERR      := Nmt.ERR;
Nmt_ERNO     := Nmt.ERNO;
```

CANOM_NODE_DIAG Polling diagnosis data from a slave



The block CANOM_NODE_DIAG reads the diagnosis data of a slave.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CANOM_NODE_DIAG	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
NODE	Input	BYTE	Node address of the concerning slave
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
STATE_1	Output	NODESTATUS_1_TYPE	Nodestatus_1, slave diagnosis
INFO	Output	WORD	Additional information according to CiA specification from the object 16#1000
PROFILE	Output	WORD	Profile number according to CiA specification from the object 16#1000
STATE	Output	BYTE	Operating condition of the slave
ONL_ERR	Output	BYTE	Online error of the slave
NUM_EMCY	Output	BYTE	Number of emergency messages in EMCY_DATA
EMCY_DATA	Output	ARRAY[1...5] OF CANOM_EMCY_TYPE	Content of the emergency messages

Description

Using the block CANOM_NODE_DIAG, diagnosis data of the individual slaves can be requested.

Every time a FALSE->TRUE edge is applied at input EN, CANOM_NODE_DIAG reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at the outputs ERR and ERNO and the termination of the request processing is indicated by DONE = TRUE.

During the processing of a request, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

NODE BYTE (node)

The node ID of the slave the diagnosis data of which are to be requested, is specified at input NODE.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

STATE_1 CANOM_NODESTATUS_1_TYPE (state 1)

STATE_1 outputs different bits as a structure of the type CANOM_NODESTATUS_1_TYPE which display error states of the slave. STATE_1 is only valid if DONE = TRUE and ERR = FALSE.

The structure of the type CANOM_NODESTATUS_1_TYPE is defined in the CANopen library (see description below).

INFO WORD (info)

Output INFO provides additional information according to the CiA specification from the object 16#1000.

PROFILE WORD (profile)

Output PROFILE provides the profile number according to the CiA specification from the object 16#1000.

STATE BYTE (state)

Output STATE provides the current operating condition of the relevant slave. STATE is only valid if DONE = TRUE and ERR = FALSE.

The following table describes the possible values of STATE and their meanings as specified in the CANopen specification.

STATE	Meaning
1	Disconnected
2	Connecting
3	Preparing
4	Prepared
5	Operational
127	Pre-Operational

ONL_ERR BYTE (online error)

ONL_ERR outputs a value describing possibly existing communication errors between the master coupler and the slave. ONL_ERR is only valid if DONE = TRUE and ERR = FALSE.

The error IDs of ONL_ERR correspond to the IDs of output CANOM_ERR.EVENT of the block CANOM_STATE. They are described in the table provided in the CANOM_STATE block description.

NUM_EMCY BYTE (number of emergency telegrams)

NUM_EMCY outputs the number of valid emergency messages of the slave output at EMCY_DATA according to the CANopen specification. Up to 5 emergency messages per slave can be buffered in the coupler.

NUM_EMCY is only valid if DONE = TRUE and ERR = FALSE.

EMCY_DATA ARRAY[1...5] OF CANOM_EMCY_TYPE (emergency data)

EMCY_DATA outputs up to 5 buffered emergency messages of the slave. The number of valid messages is output by NUM_EMCY. The structure of the type CANOM_EMCY_TYPE is defined in the CANopen library (see description below). EMCY_DATA is only valid if DONE = TRUE and ERR = FALSE.

Function call in IL

```
CAL NodeDiag (
    EN       := NodeDiag_EN,
    SLOT     := NodeDiag_SLOT
    NODE     := NodeDiag_NODE

LD  NodeDiag.DONE
ST  NodeDiag_DONE

LD  NodeDiag.ERR
ST  NodeDiag_ERR

LD  NodeDiag.ERNO
ST  NodeDiag_ERNO

LD  NodeDiag.STATE_1
ST  NodeDiag_STATE_1

LD  NodeDiag.INFO
ST  NodeDiag_INFO

LD  NodeDiag.PROFILE
ST  NodeDiag_PROFILE

LD  NodeDiag.STATE
ST  NodeDiag_STATE

LD  NodeDiag.ONL_ERR
ST  NodeDiag_ONL_ERR

LD  NodeDiag.NUM_EMCY
ST  NodeDiag_NUM_EMCY

LD  NodeDiag.EMCY_DATA
ST  NodeDiag_EMCY_DATA
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
NodeDiag(  EN       := NodeDiag_EN,
           SLOT     := NodeDiag_SLOT,
           NODE     := NodeDiag_NODE;

NodeDiag_DONE      := NodeDiag.DONE;
NodeDiag_ERR       := NodeDiag.ERR;
NodeDiag_ERNO      := NodeDiag.ERNO;
NodeDiag_STATE_1   := NodeDiag.STATE_1;
NodeDiag_INFO      := NodeDiag.INFO;
NodeDiag_PROFILE   := NodeDiag.PROFILE;
NodeDiag_STATE     := NodeDiag.STATE;
NodeDiag_ONL_ERR   := NodeDiag.ONL_ERR;
NodeDiag_NUM_EMCY  := NodeDiag.NUM_EMCY;
NodeDiag_EMCY_DATA := NodeDiag.EMCY_DATA;
```

STATE_1   CANOM_NODESTATUS_1_TYPE

Output STAT_1 of the block CANOM_NODE_DIAG displays different diagnosis bits as a structure of the type CANOM_NODESTATUS_1_TYPE. The structure CANOM_NODESTATUS_1_TYPE is declared as follows in the CANopen library:

```
TYPE CANOM_NODESTATUS_1_TYPE:
STRUCT
    NO_RESPONSE: BOOL;
    EMCY_OVF:    BOOL;
    PRM_FAULT:   BOOL;
    GUARD_ACT:   BOOL;
    reserved1:   BOOL;
    reserved2:   BOOL;
    reserved3:   BOOL;
    DEACTIVATED: BOOL;
END_STRUCT
END_TYPE
```

NO_RESPONSE   BOOL   (no response)

If this bit is set, the slave with the node number specified at block input NODE does not respond to the master requests. Normally NO_RESPONSE should be set to FALSE.

EMCY_OVF   BOOL   (emergency overflow)

This bit is set by the coupler, if more emergency messages were received from the called slave than the buffer can store (refer to block outputs NUM_EMCY and EMCY_DATA).

PRM_FAULT   BOOL   (parameter fault)

This bit is set, if the nominal slave configuration defined in the master differs from the actual slave configuration.

GUARD_ACT   BOOL   (guarding active)

This bit is set by the coupler, if the node guarding protocol for this slave is active. This is only a status indication. The active node guarding protocol between the master and the slave is not synonymous with a node guarding error.

reserved1   BOOL

reserved2   BOOL

reserved3   BOOL

These bits are reserved and are currently not in use.

DEACTIVATED   BOOL   (deactivated)

This bit is set to TRUE, if the slave defined in the configuration data of the master is deactivated and not processed.

EMCY_DATA CANOM_EMCY_TYPE

EMCY_DATA outputs the up to 5 (refer to NUM_EMCY) emergency messages received last from the slave. EMCY_DATA consists of one ARRAY [1..5] OF CANOM_EMCY_TYPE. The data type CANOM_EMCY_TYPE corresponds to the format of the emergency telegram described in the CANopen communication profile and is defined as follows in the CANopen library:

```
TYPE CANOM_EMCY_TYPE:
STRUCT
    ERROR_CODE: WORD;
    ERROR_REG:  BYTE;
    ERROR_DATA: ARRAY[1...5] OF BYTE;
END_STRUCT
END_TYPE
```

ERROR_CODE WORD (error code)

For the emergency object the emergency error codes described in the following table are defined in the CANopen communication profile.

Emergency error code		Meaning / error cause
decimal	hexadecimal	
00000...00255	0000...00FF	Error on reset or no error
04096...04351	1000...10FF	General error
08192...08447	2000...20FF	Current error
08448...08703	2100...21FF	- error on the device input side
08704...08959	2200...22FF	- error inside the device
08960...09215	2300...23FF	- error on the device output side
12288...12543	3000...30FF	Voltage error
12544...12799	3100...31FF	- supply voltage error
12800...13055	3200...32FF	- error inside the device
13056...13311	3300...33FF	- error on the device output side
16384...16639	4000...40FF	Temperature error
16640...16895	4100...41FF	- ambient temperature
16896...17151	4200...42FF	- temperature inside the device
20480...20735	5000...50FF	Hardware error in the device
24576...24831	6000...60FF	Software error in the device
24832...25087	6100...61FF	- device-internal software
25088...25343	6200...62FF	- application software
25344...25599	6300...63FF	Data
28672...28927	7000...70FF	Error in additional modules
32768...33023	8000...80FF	- monitoring
33024...33279	8100...81FF	- communication
36864...37119	9000...90FF	External error
61440...61695	F000...F0FF	Error of additional functions
65280...65535	FF00...FFFF	Device-specific errors

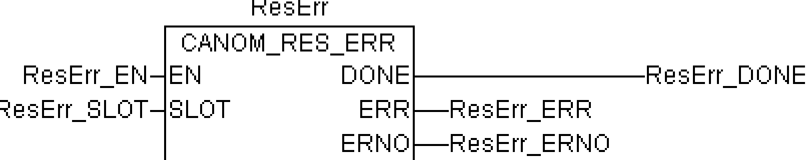
ERROR_REG BYTE (error register)

The variable ERROR_REG displays the error register value (object 1001 hex) of the slave. This value is transmitted by the slave as a part of the emergency message.

ERROR_DATA ARRAY[1...5] OF BYTE (error data)

If applicable, ERROR_DATA is used to output manufacturer-specific error information transmitted by the slave as part of the emergency message. For detailed information about the meaning of these data please refer to the particular device documentation.

CANOM_RES_ERR Resetting the coupler's error indications



The block CANOM_RES_ERR can be used to reset various internal error indications and counters of the coupler.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CANOM_RES_ERR	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

Using the block CANOM_RES_ERR it is possible to reset the following coupler internal error indications and error counters output via the block CANOM_STAT:

- STATE_BITS.EVENT
- STATE_BITS.TIMEOUT
- BUS_ERR
- BUS_OFF
- TOUT_ERR
- LOST_REC

For explanations of the error indications please refer to the description of the block CANOM_STAT.

The reset is initiated by a FALSE->TRUE edge at input EN.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at the outputs ERR and ERNO and the termination of the request processing is indicated by DONE = TRUE.

During the processing of a request, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

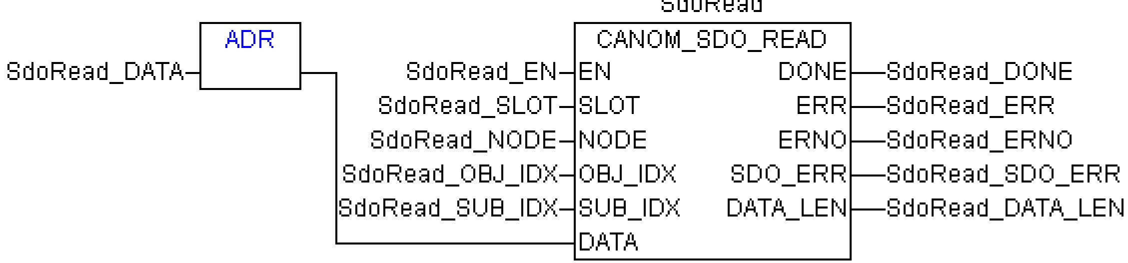
```
CAL ResErr (
    EN      := ResErr_EN,
    SLOT    := ResErr_SLOT)

LD  ResErr.DONE
ST  ResErr_DONE

LD  ResErr.ERR
ST  ResErr_ERR

LD  ResErr.ERNO
ST  ResErr_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
ResErr( EN     := ResErr_EN,
        SLOT   := ResErr_SLOT);

ResErr_DONE    := ResErr.DONE;
ResErr_ERR     := ResErr.ERR;
ResErr_ERNO    := ResErr.ERNO;
```

CANOM_SDO_READ Reading the value of a slave object



The block CANOM_SDO_READ can be used to read individual service data objects (SDOs) from a slave.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CANOM_SDO_READ	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
NODE	Input	BYTE	Node address of the concerning slave
OBJ_IDX	Input	WORD	Object index of the object to be read
SUB_IDX	Input	BYTE	Sub index of the object to be read
DATA	Input	DWORD	Address from which on the read data are stored (via ADR operator)
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
SDO_ERR	Output	DWORD	SDO abort error message of the slave
DATA_LEN	Output	BYTE	Length of read data (byte value)

Description

Using the block CANOM_SDO_READ it is possible to read service data objects (SDOs) from a slave.

Every time a FALSE->TRUE edge is applied to input EN, CANOM_SDO_READ reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

The CANopen object model is defined in the communication profile and device profile specifications (see also System technology of the CANopen couplers). Furthermore, the device-specific objects are explained in the corresponding device description of the slave.

Some objects have to be available in every CANopen device by default. Normally, these standard objects are not accessed by the user program during running operation, even though this is possible in principle. Access is usually restricted to additional optional objects or their values. These additional slave attributes can be viewed in SYCON.net by selecting Device configuration / SDO table in the CANopen Master (see also the documentation of the DTM fieldbus configurator for CANopen devices). This view displays a list of predefined objects for the corresponding slave gathered from the EDS file, and an overview of those standard objects that are automatically written by the master during the startup sequence of the node. Each entry includes the running object index, the sub index, a description of the parameter, the parameter's default value and its actual value as well as the permitted access methods (read/write).

This information can be used to select the desired OBJ_IDX and SUB_IDX when using a function block of the type CANOM_SDO_READ.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at the outputs ERR and ERNO and the termination of the request processing is indicated by DONE = TRUE.

During the processing of a request, state changes at input EN are recognized but not evaluated. Since block execution requires bus access, the data are available in the next cycle after activating the block at the earliest.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

NODE BYTE (node)

At input NODE the node address of the slave is specified the object value of which is to be requested.

OBJ_IDX WORD (object index)

Input OBJ_IDX is used to specify the object index of the object to be read from the slave object directory (compare to entry Obj. Idx. listed in SYCON.net when using the option Device configuration / SDO table).

SUB_IDX BYTE (subject index)

Input SUB_IDX is used to specify the sub index of the object to be read from the slave object directory (compare to entry Sub. Idx. listed in SYCON.net when using the option Device configuration / SDO table).

DATA DWORD (data)

The address of the variable to which the received object data are to be written is specified at input DATA via the ADR operator. Using the ADR operator, a variable of any data format can be specified according to the format of the object applied at input DATA. It is absolutely necessary that the size of the variables (e.g. ARRAY) is sufficient to hold the amount of read data. The received data are only valid if DONE = TRUE, ERR = FALSE and DATA_LEN > 0.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

If the slave answered the SDO message with an abort message, this is displayed by ERNO = 6003 hex (24579 dec). In many cases this is the result of an invalid object index or sub index or of missing access rights. In this case, the error code transmitted by the slave is additionally output at SDO_ERR.

SDO_ERR DWORD (SDO error)

If the slave answered the SDO message with an abort message, the transmitted error code is output at SDO_ERR. The value output at SDO_ERR is only valid if DONE = TRUE, ERR = TRUE and ERNO = 6003hex (24579dec).

DATA_LEN BYTE (data length)

DATA_LEN outputs the length of the received object data (in bytes), after the procedure has been completed successfully. Since block execution requires bus access, the data are available in the next cycle after activating the block at the earliest. The output value is only valid, if DONE = TRUE and ERR = FALSE.

Function call in IL

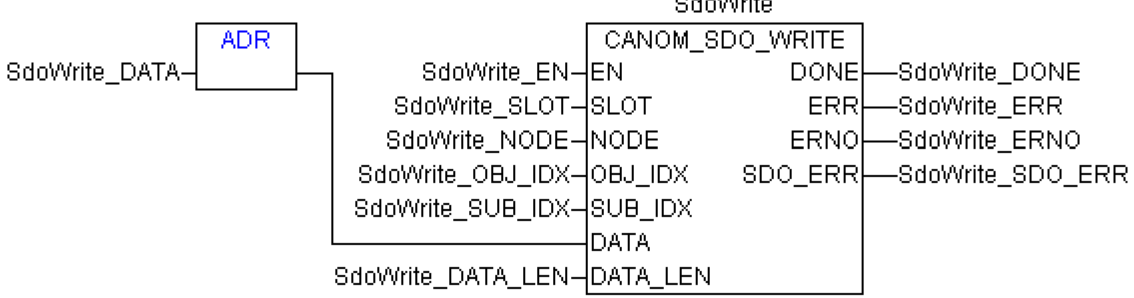
```
LD     SdoRead_DATA
ADR
ST     ADR_SdoRead_DATA

CAL    SdoRead (
       EN      := SdoRead_EN,
       SLOT    := SdoRead_SLOT
       NODE    := SdoRead_NODE
       OBJ_IDX := SdoRead_OBJ_IDX
       SUB_IDX := SdoRead_SUB_IDX
       DATA    := ADR_SdoRead_DATA)

LD     SdoRead.DONE
ST     SdoRead_DONE

LD     SdoRead.ERR
ST     SdoRead_ERR

LD     SdoRead.ERNO
ST     SdoRead_ERNO

LD     SdoRead.SDO_ERR
ST     SdoRead_SDO_ERR

LD     SdoRead.DATA_LEN
ST     SdoRead_DATA_LEN
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
SdoRead ( EN      := SdoRead_EN,
          SLOT    := SdoRead_SLOT,
          NODE    := SdoRead_NODE,
          OBJ_IDX := SdoRead_OBJ_IDX
          SUB_IDX := SdoRead_SUB_IDX
          DATA    := ADR(SdoRead_DATA);

SdoRead_DONE      := SdoRead.DONE;
SdoRead_ERR       := SdoRead.ERR;
SdoRead_ERNO      := SdoRead.ERNO;
SdoRead_SDO_ERR   := SdoRead.SDO_ERR;
SdoRead_DATA_LEN  := SdoRead.DATA_LEN;
```

CANOM_SDO_WRITE Writing the value of a slave object



The block CANOM_SDO_WRITE can be used to write individual service data objects (SDOs) to a slave.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CANOM_SDO_WRITE	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
NODE	Input	BYTE	Node address of the concerning slave
OBJ_IDX	Input	WORD	Object index of the object to be written
SUB_IDX	Input	BYTE	Sub index of the object to be written
DATA	Input	DWORD	Address from which on the written data are stored (via ADR operator)
DATA_LEN	Input	BYTE	Length of data to be written (byte value)
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
SDO_ERR	Output	DWORD	SDO abort error message of the slave

Description

The block CANOM_SDO_WRITE can be used to write service data objects (SDOs) of a slave.

Every time a FALSE->TRUE edge is applied to input EN, CANOM_SDO_WRITE reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

The CANopen object model is defined in the communication profile and device profile specifications (see also System technology of CANopen couplers). Furthermore, the device-specific objects are explained in the corresponding device description of the slave.

Some SDOs have to be available in every CANopen device by default. Normally, these standard objects are not accessed by the user program during running operation, even though this is possible in principle. Access is usually restricted to additional optional objects or their values. These additional slave attributes can be viewed in SYCON.net by selecting Device configuration / SDO table in the CANopen Master (see also the documentation of the DTM fieldbus configurator for CANopen devices). This view displays a list of predefined objects for the corresponding slave gathered from the EDS file, and an overview of those standard objects that are automatically written by the master during the startup sequence of the node. Each entry includes the running object index, the sub index, a description of the parameter, the parameter's default value and its actual value as well as the permitted access methods (read/write). This information can be used to select the desired OBJ_IDX and SUB_IDX when using a function block of the type CANOM_SDO_WRITE.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at the outputs ERR and ERNO and the termination of the request processing is indicated by DONE = TRUE.

During the processing of a request, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

NODE BYTE (node)

At input NODE the node address of the slave is specified the object value of which is to be written.

OBJ_IDX WORD (object index)

Input OBJ_IDX is used to specify the object index of the object to be written to the slave object directory (compare to entry Obj. Idx. listed in SYCON.net when using the option Device configuration / SDO table).

SUB_IDX BYTE (subject index)

Input SUB_IDX is used to specify the sub index of the object to be written to the slave object directory (compare to entry Sub. Idx. listed in SYCON.net when using the option Device configuration / SDO table).

DATA DWORD (data)

The address of the variable containing the object data to be transmitted is specified at input DATA via the ADR operator. Using the ADR operator, a variable of any data format can be specified according to the format of the object applied at input DATA.

DATA_LEN BYTE (data length)

At input DATA_LEN, the length of the data to be transmitted stored in the variable at address DATA is specified as a byte value. This value must match the size of the object to be written. The maximum data length is 247 bytes.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

If the slave answered the SDO message with an abort message, this is displayed by ERR = 6003hex (24579dec). In many cases this is the result of an invalid object index or sub index or of missing access rights. In this case, the error code transmitted by the slave is additionally output at SDO_ERR.

SDO_ERR DWORD (SDO error)

If the slave answered the SDO message with an abort message, the transmitted error code is output at SDO_ERR. The value output at SDO_ERR is only valid if DONE = TRUE, ERR = TRUE and ERNO = 6003hex (24579dec).

Function call in IL

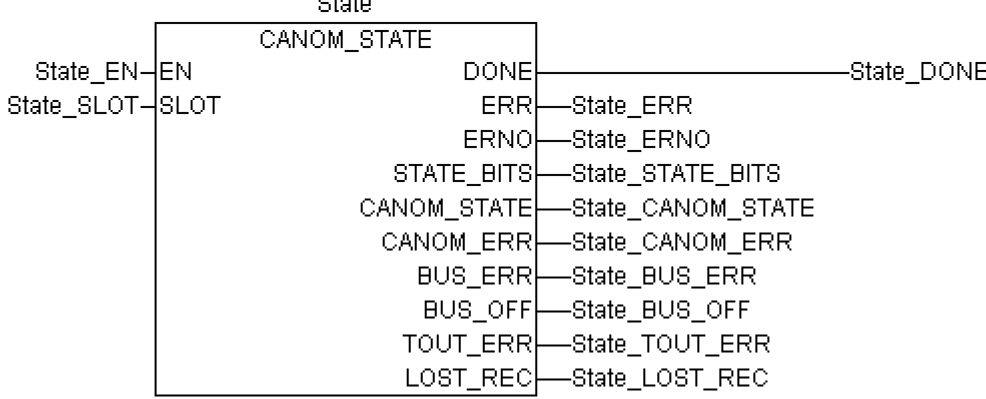
```
LD     SdoWrite_DATA
ADR
ST     ADR_SdoWrite_DATA

CAL    SdoWrite (
       EN       := SdoWrite_EN,
       SLOT     := SdoWrite_SLOT
       NODE     := SdoWrite_NODE
       OBJ_IDX  := SdoWrite_OBJ_IDX
       SUB_IDX  := SdoWrite_SUB_IDX
       DATA     := ADR_SdoWrite_DATA
       DATA_LEN := ADR_SdoWrite_DATA_LEN)

LD     SdoWrite.DONE
ST     SdoWrite_DONE

LD     SdoWrite.ERR
ST     SdoWrite_ERR

LD     SdoWrite.ERNO
ST     SdoWrite_ERNO

LD     SdoWrite.SDO_ERR
ST     SdoWrite_SDO_ERR
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
SdoWrite ( EN       := SdoWrite_EN,
           SLOT     := SdoWrite_SLOT,
           NODE     := SdoWrite_NODE,
           OBJ_IDX  := SdoWrite_OBJ_IDX,
           SUB_IDX  := SdoWrite_SUB_IDX,
           DATA     := ADR(SdoWrite_DATA,
           DATA_LEN := ADR(SdoWrite_DATA_LEN);

SdoWrite_DONE       := SdoWrite.DONE;
SdoWrite_ERR        := SdoWrite.ERR;
SdoWrite_ERNO       := SdoWrite.ERNO;
SdoWrite_SDO_ERR    := SdoWrite.SDO_ERR;
```

CANOM_STATE Reading the CANopen coupler status



CANOM_STATE outputs the status of a CANopen coupler. The outputs provide information about the communication status and error events.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CANOM_STATE	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
STATE_BITS	Output	CANOM_STATE_BITS_TYPE	Atypical communication states
CANOM_STATE	Output	BYTE	General state of the CANopen master
CANOM_ERR	Output	CANOM_COM_ERR_TYPE	Communication error
BUS_ERR	Output	WORD	Number of bus errors
BUS_OFF	Output	WORD	Number of bus outages
TOUT_ERR	Output	WORD	Number of dismissed transmitted CAN messages
LOST_REC	Output	WORD	Number of dismissed received CAN messages

Description

The block CANOM_STATE outputs the current status of the CANopen coupler.

CANOM_STATE is active if input EN = TRUE. If the block is active and if no errors occurred during block processing, the current values are permanently displayed at the outputs.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

STATE_BITS CANOM_STATE_BITS_TYPE (state bits)

Output STATE_BITS indicates atypical communication states of the CANopen coupler. STATE_BITS is only valid if EN = TRUE and ERR = FALSE.

The structure of the type CANOM_STATE_BITS_TYPE is defined in the CANopen library (see description below).

Some of the error indications in STATE_BITS can be reset using the block CANOM_RES_ERR.

CANOM_STATE BYTE (CANopen master state)

CANOM_STATE outputs the general communication state of the CANopen master. The following states are defined:

State		Meaning
Dec	Hex	
0	00	OFFLINE
64	40	STOP
128	80	CLEAR
192	C0	OPERATE

CANOM_STATE = OFFLINE

If CANOM_STATE is set to OFFLINE, the CANopen coupler performs an initialization. After the initialization phase is completed, the coupler changes to STOP state.

CANOM_STATE = STOP

If CANOM_STATE has the value STOP, the coupler is completely initialized. In this state the coupler is ready to receive configuration data. There is no data exchange with the slaves. The coupler has this state if no user program is running.

CANOM_STATE = CLEAR

If the user program is started, the coupler changes from STOP to CLEAR and starts to establish the connections defined during configuration. When the setup has been completed successfully, the coupler moves to OPERATE state. If an error occurs during parameterization, the coupler changes back to STOP state.

CANOM_STATE = OPERATE

Normally, the coupler is in OPERATE state while a user program is running. In this state the master exchanges I/O data with the slaves. If an error occurs during this process and if 'Auto Clear Mode' has been selected during configuration, the coupler changes back to CLEAR state and tries to establish the connections again. If 'Auto Clear Mode' has not been selected during configuration, the coupler remains in OPERATE state in case of an error. If the user program is stopped, the coupler also changes back to STOP state.

CANOM_STATE is only valid, if EN = TRUE and ERR = FALSE.

CANOM_ERR CANOM_COM_ERR_TYPE (CANopen master error)

Output CANOM_ERR indicates possibly occurring communication errors. CANOM_ERR is only valid, if EN = TRUE and ERR = FALSE.

The structure of the type CANOM_COM_ERR_TYPE is defined in the CANopen library and described below together with the possible errors.

BUS_ERR WORD (bus error)

BUS_ERR outputs the number of occurred bus failures. A bus failure occurs if the internal error frame counter exceeds a specific value. BUS_ERR is only valid if EN = TRUE and ERR = FALSE.

BUS_ERR can be reset using the block CANOM_RES_ERR.

BUS_OFF WORD (bus off)

BUS_OFF outputs how often the coupler has been excluded from bus activities. An exclusion from the bus activities is performed, if an overflow of the internal error frame counter occurs. The coupler is automatically re-initialized after each overflow. BUS_OFF is only valid, if EN = TRUE and ERR = FALSE.

BUS_OFF can be reset using the block CANOM_RES_ERR.

TOUT_ERR WORD (timeout error)

TOUT_ERR outputs the number of telegrams that could not be transmitted successfully. The transmission of a telegram is considered as failed, if it could not be transmitted within 20 ms, for instance because the communication partner could not be contacted via the bus. TOUT_ERR is only valid, if EN = TRUE and ERR = FALSE.

TOUT_ERR can be reset using the block CANOM_RES_ERR.

LOST_REC WORD (lost receive)

LOST_REC outputs the number of received telegrams that were rejected because they could not be processed successfully due to a CAN chip overload. LOST_REC is only valid, if EN = TRUE and ERR = FALSE.

LOST_REC can be reset using the block CANOM_RES_ERR.

Function call in IL

```
CAL    State (
       EN         := State_EN,
       SLOT       := State_SLOT)

LD     State.DONE
ST     State_DONE

LD     State.ERR
ST     State_ERR

LD     State.ERNO
ST     State_ERNO

LD     State.STATE_BITS
ST     State_STATE_BITS

LD     State.CANOM_STATE
ST     State_CANOM_STATE

LD     State.CANOM_ERR
ST     State_CANOM_ERR

LD     State.BUS_ERR
ST     State_BUS_ERR

LD     State.BUS_OFF
ST     State_BUS_OFF

LD     State.TOUT_ERR
ST     State_TOUT_ERR

LD     State.LOST_REC
ST     State_LOST_REC
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
State ( EN          := State_EN,
        SLOT        := State_SLOT);

State_DONE          := State.DONE;
State_ERR           := State.ERR;
State_ERNO          := State.ERNO;
State_STATE_BITS    := State.STATE_BITS;
State_CANOM_STATE   := State.CANOM_STATE;
State_CANOM_ERR     := State.CANOM_ERR;
State_BUS_ERR       := State.BUS_ERR;
State_BUS_OFF       := State.BUS_OFF;
State_TOUT_ERR      := State.TOUT_ERR;
State_LOST_ERR      := State.LOST_ERR;
```

STATE_BITS CANOM_STATE_BITS_TYPE

The structure STATE_BITS consists of six boolean variables displaying different communication states. In the CANopen library, the data type CANOM_STATE_BITS_TYPE is defined as follows:

```
TYPE CANOM_STATE_BITS_TYPE:
STRUCT
   CTRL:       BOOL;
   AUTO_CLR:   BOOL;
   NO_EXCH:    BOOL;
   FATAL:      BOOL;
   EVENT:      BOOL;
   reserved1: BOOL;
   TIMEOUT:    BOOL;
   reserved2: BOOL;
END_STRUCT
END_TYPE
```

CTRL BOOL (control)

If this bit is TRUE, a parameter setting error occurred. During normal operation, CTRL should be FALSE. If this is not the case, the parameter and configuration data have to be checked.

AUTO_CLR BOOL (auto clear)

If AUTO_CLR is set to TRUE, the coupler stopped data exchange with all slaves due to communication errors and changed back to CLEAR state (see also CANOM_STATE).

NO_EXCH BOOL (no exchange)

This bit is set to TRUE, if no exchange of process data can be performed with one or several slaves. The error can be caused by the configuration data or the slaves themselves.

FATAL BOOL (fatal)

If FATAL is set to TRUE, no communication via CANopen is possible due to a fatal internal error.

EVENT BOOL (event)

EVENT is set to TRUE, if the coupler detects transmission errors. The number of occurring transmission errors is displayed at the corresponding outputs BUS_ERR and BUS_OFF. If the EVENT bit is set to TRUE, reset is only possible via the block CANOM_RES_ERR.

reserved1 BOOL

This bit is reserved and currently not in use.

TIMEOUT BOOL (timeout)

If TIMEOUT is set to TRUE, transmission of at least one telegram failed. The transmission of this telegram was aborted and its content is lost. TIMEOUT = TRUE is an indication that the communication partner could not be contacted via the bus. The number of failed transmissions is displayed at output TOUT_ERR. If the TIMEOUT bit is set to TRUE, reset is only possible using the block CANOM_RES_ERR.

reserved2 BOOL

This bit is reserved and currently not in use.

CANOM_ERR   CANOM_COM_ERR_TYPE

Using CANOM_ERR, it is possible to locate communication errors more precisely. The output CANOM_ERR is represented as a structure of the type CANOM_COM_ERR_TYPE. In the CANopen library this data type is declared as follows:

```
TYPE CANOM_COM_ERR_TYPE:
STRUCT
   ADDRESS: BYTE;
   EVENT:   BYTE;
END_STRUCT
END_TYPE
```

ADDRESS   BYTE   (address)

In case of an error, ADDRESS contains the node address of the faulty device. If ADDRESS has the value 255, the error is located in the coupler itself.

EVENT   BYTE   (event)

In case of an error, EVENT contains the error causing event. The event refers to a single node address (ADDRESS <> 255) or the coupler itself (ADDRESS = 255).

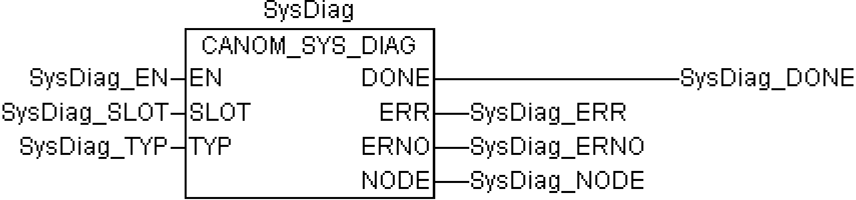
ADDRESS <> 255 Error at subscribers with node address ADDRESS

Event	Meaning	Error source	Cause / Remedy
30	Slave monitoring timeout	Slave	Check connection and node address of the slave
31	Slave aborted operational mode	Slave	Reset slave
32	Sequence error in node guarding protocol	Slave	Reset slave
33	No response to configured remote frame PDO	Slave	Check whether slave remote frames are supported
34	No slave response during configuration	Slave	Check whether the slave is connected and ready for operation
35	Actual device profile of the slave differs from the configured device profile	Configuration	Check the profile supported by the slave
36	Actual device type of the slave differs from the configured device type	Configuration	Check the services supported by the slave
37	Unknown SDO response received	Slave	Slave does not meet the CiA protocol specifications
38	Length indicator of a received SDO response is not equal to 8	Slave	Slave does not meet the CiA protocol specifications
39	Slave is not processed. It is in the STOP state.	Configuration/ Coupler	Deactivate the 'Auto Clear Mode'

ADDRESS = 255 Coupler error

Event	Meaning	Error source	Cause / Remedy
52	Unknown process data handshake mode	Configuration	Contact customer support
56	Invalid data transfer rate	Configuration	Contact customer support
60	Node address configured twice	Configuration / other device	Check node addresses of all devices specified in configuration data
210	No configuration data	Configuration / Coupler	Load configuration data into coupler
212	Error while reading database	Configuration / Coupler	Load configuration data into coupler again, contact customer support
220	Watchdog error	Controller	Contact customer support

CANOM_SYS_DIAG Displaying status surveys of all slaves



The block CANOM_SYS_DIAG outputs a bit field as a state survey of all slaves (nodes) at output NODE. Three different survey types can be selected via input TYP.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	CANopen_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CANOM_SYS_DIAG	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
TYP	Input	BYTE	Selection of the survey type
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NODE	Output	ARRAY[0...127] OF BOOL	Status survey of the slaves

Description

The block CANOM_SYS_DIAG outputs different status surveys of all slaves. Three survey types can be selected:

- configuration survey
- operational survey
- diagnosis survey

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

SLOT   BYTE   (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

TYP   BYTE   (type)

The input TYP is used to select the type of status survey displayed at output NODE.

TYP = 1 Configuration survey

Output NODE displays which slaves are successfully connected to the master (TRUE). Please note that the master only establishes connections to slaves that have been announced to the master during the definition of the configuration data.

TYP = 2 Operational survey

Output NODE displays which slaves are error-free and in operation. A slave can only be announced as operational if it has been configured in the master. The operational survey can only be requested if the coupler is in OPERATE state.

TYP = 3 Diagnosis survey

Output NODE indicates which slaves report a diagnosis. The diagnosis survey can only be requested if the coupler is in OPERATE state.

DONE   BOOL   (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR   BOOL   (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO   WORD   (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

NODE   ARRAY[0...127] OF BOOL   (node)

NODE outputs the status survey as a bit field. Each individual bit within this field represents one slave. The index number corresponds to the slave's node address. If a bit is set to TRUE, the state selected using TYP applies to the corresponding slave.

If e.g. TYP = 1 is selected and NODE[2] = TRUE, the slave with this node address was successfully configured by the master and is currently in operation. If NODE[2] = FALSE, the configuration of the specific slave has not yet been completed or the slave is not part of the master's configuration data.

If TYP = 3, NODE[2] = TRUE for example means that the slave with the node address 2 has received an emergency message or that the diagnosis indication of the slave has changed. In this case, a detailed diagnosis can be requested using the block CANOM_NODE_DIAG.

The bit field output at NODE is only valid, if EN = TRUE and ERR = FALSE.

Function call in IL

```
CAL SysDiag (
    EN      := SysDiag_EN,
    SLOT    := SysDiag_SLOT,
    TYP     := SysDiag_TYP)

LD  SysDiag.DONE
ST  SysDiag_DONE

LD  SysDiag.ERR
ST  SysDiag_ERR

LD  SysDiag.ERNO
ST  SysDiag_ERNO

LD  SysDiag.NODE
ST  SysDiag_NODE
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
SysDiag( EN    := SysDiag_EN,
         SLOT  := SysDiag_SLOT,
         TYP   := SysDiag_TYP);

SysDiag_DONE   := SysDiag.DONE;
SysDiag_ERR    := SysDiag.ERR;
SysDiag_ERNO   := SysDiag.ERNO;
SysDiag_NODE   := SysDiag.NODE;
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

CAN2A INFO Reading information about CAN 2.0A communication 6

CAN2A_MESSAGE_TYPE Telegram structure according to CAN 2.0A 3

CAN2A_REC Reading CAN 2.0A telegrams (with 11 bit identifier) from a receive buffer 9

CAN2A_SEND Transmitting CAN 2.0A telegrams (with 11 bit identifier) 13

CAN2B_INFO Reading information about CAN 2.0B communication 16

CAN2B_MESSAGE_TYPE Telegram structure according to CAN 2.0B 4

CAN2B_REC Reading CAN 2.0B telegrams (with 29 bit identifier) from a receive buffer 19

CAN2B_SEND Transmitting CAN 2.0B telegrams (with 29 bit identifier) 23

CANOM_NMT Controlling NMT node states via network management 26

CANOM_NODE_DIAG Polling diagnosis data from a slave 29

CANOM_RES_ERR Resetting the coupler's error indications 36

CANOM_SDO_READ Reading the value of a slave object 39

CANOM_SDO_WRITE Writing the value of a slave object 43

CANOM_STATE Reading the CANopen coupler status 47

CANOM_SYS_DIAG Displaying status surveys of all slaves 54

Components of the CANopen library 2

G

General notes regarding CAN blocks 3

Glossary 57

O

Overview of blocks arranged according to their call names 5

S

Special characteristics of the CANopen library 2

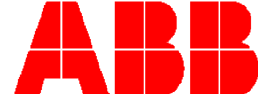
Software Description

AC500

Scalable PLC
for Individual Automation

DeviceNet
Function Block Library

DeviceNet



Contents

DeviceNet Library 2

Special characteristics of the DeviceNet Library 2

Components of the DeviceNet Library 2

Overview of blocks arranged according to their call names 3

DNM_DEV_DIAG Polling diagnosis data from a slave 4

DNM_GET_ATTR Reading an attribute from a slave object 10

DNM_RES_OBJ Resetting a slave object 14

DNM_SET_ATTR Writing an attribute to a slave object 17

DNM_STATE Reading the DeviceNet coupler status 21

DNM_SYS_DIAG Displaying status surveys of all slaves 28

Glossary 31

Index 33

DeviceNet Library

Special characteristics of the DeviceNet library

Note:

DeviceNet communication is only performed in RUN mode of the PLC and not in simulation mode.

Prerequisites for using the library

The function blocks contained in the DeviceNet library access to both, the PLC run time system and directly to the coupler. The definitions and functions required for this are stored in the internal library SysExt_AC500_V10.LIB. This library is automatically included during project setup.

The library contains function blocks for a comfortable handling of the DeviceNet coupler as well as definitions of various data types. These structures enable a clear presentation of data sets.

Components of the DeviceNet library

Function blocks

The DeviceNet library contains the following function blocks:

Group: Diagnosis	
DNM_DEV_DIAG	Polling diagnosis data from a slave
DNM_STATE	Reading the DeviceNet coupler status
DNM_SYS_DIAG	Displaying status information of all slaves

Group: Parameters	
DNM_GET_ATTR	Reading an attribute from a slave object
DNM_RES_OBJ	Resetting a slave object
DNM_SET_ATTR	Writing an attribute to a slave object

Data types

The following data types (structures) are defined in the DeviceNet library:

Group: DNM	
DNM_COM_ERR_TYPE	Communication error
DNM_DEVICESTATUS_1_TYPE	Slave diagnosis
DNM_STATE_BITS_TYPE	Bits for coupler state description

The different data types are described in the documentation of the respective block.

Overview of blocks arranged according to their call names

Character description:

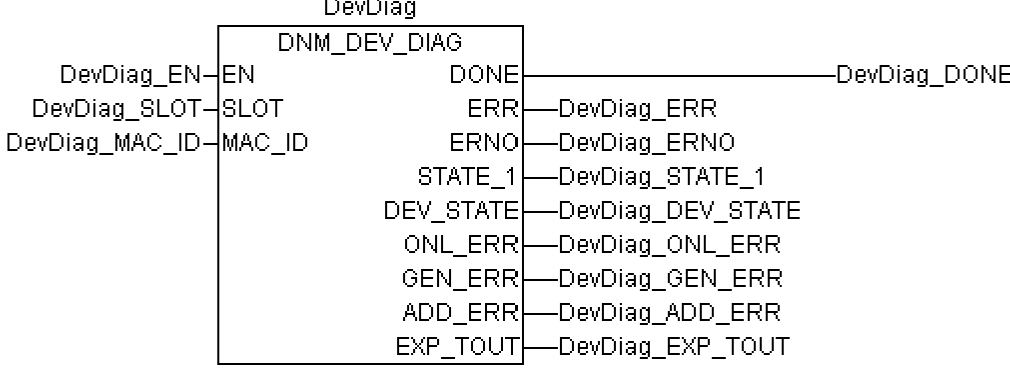
FBhV … Function block with historical values

FBnohv … Function block block without historical values

F … Function

VE name	Type	Function
DNM_DEV_DIAG	FBhv	Polling diagnosis data from a slave
DNM_GET_ATTR	FBhv	Reading an attribute from a slave object
DNM_RES_OBJ	FBhv	Resetting a slave object
DNM_SET_ATTR	FBhv	Writing an attribute to a slave object
DNM_STATE	FBhv	Reading the DeviceNet coupler status
DNM_SYS_DIAG	FBhv	Displaying status surveys of all slaves

DNM_DEV_DIAG Polling diagnosis data from a slave



The block DNM_DEV_DIAG reads the diagnosis data of a slave.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	DeviceNet_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DNM_DEV_DIAG	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
MAC_ID	Input	BYTE	MAC ID of the relevant slave
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
STATE_1	Output	DNM_DEVICESTATUS_1_TYPE	Devicestatus_1, slave diagnosis
DEV_STATE	Output	BYTE	Status of the slave state automation
ONL_ERR	Output	BYTE	Online error of the slave
GEN_ERR	Output	BYTE	General error of the slave
ADD_ERR	Output	BYTE	Additional error information
EXP_TOUT	Output	WORD	ExpectedPacketRate_Timeout

Description

Using the block DNM_DEV_DIAG, diagnosis data of individual slaves can be requested.

Every time a FALSE->TRUE edge is applied to input EN, DNM_DEV_DIAG reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at the outputs ERR and ERNO and the termination of the request processing is indicated by DONE = TRUE.

During the processing of a request, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

MAC_ID BYTE (medium access control identifier)

Input MAC_ID is used to specify the MAC ID of the slave, the diagnosis data of which are to be requested.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

STATE_1 DNM_DEVICESTATUS_1_TYPE (state 1)

STAT_1 outputs different bits as a structure of the type DNM_DEVICESTATUS_1_TYPE to display the error states of the slave. STAT_1 is only valid if DONE = TRUE and ERR = FALSE.

The structure of the type DNM_DEVICESTATUS_1_TYPE is defined in the DeviceNet library (see description below).

DEV_STATE BYTE (device state)

DEV_STATE outputs the current state of the coupler's internal state machine for the relevant slave. This value provides information about the current phase of communication with the device. DEV_STATE is only valid if DONE = TRUE and ERR = FALSE.

The following table lists the possible values of DEV_STATE and their meanings.

DEV_STATE	Meaning
1	Initial state of the state automation
2	Slave inactive and not processed
3	Waiting for requests of all slaves to check for duplicate MAC IDs
4	Initialization of the internal predefined master-slave structures
5	Request for allocation of predefined master-slave connection sets
6	Waiting for slave response after request for allocation of predefined master-slave connection sets
7	Request for release of predefined master-slave connection sets
8	Waiting for slave response after request for release of predefined master-slave connection sets
9	Initialization of the internal I/O configuration structures
10	Request for allocation of I/O configuration structures
11	Waiting for slave response after request for allocation of I/O configuration structures
12	Request for release of I/O configuration structures
13	Waiting for slave response after request for release of I/O configuration structures
14	Request for connection length of the connection consumed by the slave (output configuration)
15	Waiting for slave response after request for connection length of the connection consumed by this slave
16	Comparison of actual output configuration received from the slave with the internal nominal output configuration
17	Request for connection length of the connection produced by the slave (input configuration)
18	Waiting for slave response after request for connection length of the connection produced by this slave
19	Comparison of actual input configuration received from the slave with the internal nominal input configuration
20	Configuration and registration of I/O connection structures
21	Setting the expected packet rate in the slave
22	Waiting for slave response after setting the expected packet rate
23	Transmission of the first I/O poll request to the slave
24	Waiting for slave response after I/O poll request
25	Transmission of the second I/O poll request to the slave
26	Waiting for slave response after I/O poll request
27	Transmission of the third I/O poll request to the slave
28	Waiting for slave response after I/O poll request
29	Transmission of heartbeat timeout to slave
30	-
31	First opening of connectionless communication
32	Waiting for slave response after opening the connectionless communication
33	Second opening of connectionless communication
34	Waiting for slave response after opening the connectionless communication
35	Closing the connectionless communication
36	Waiting for slave response after closing the connectionless communication
37	Enabling of all established connections
38	Waiting for slave response after enabling of all established connections
39	Opening the connectionless user-specific communication

40	Waiting for slave response after opening the connectionless user-specific communication
41	Request for allocation of user-specific predefined master-slave connections
42	Waiting for slave response after request for allocation of user-specific master-slave connections
43	Closing the connectionless user-specific communication
44	Waiting for slave response after request for closing the user-specific master-slave connections
45	Reading or writing a user-specific attribute
46	Waiting for response after reading or writing a user-specific attribute
47	Transmission or waiting for response after fragmentally reading or writing attributes

ONL_ERR BYTE (online error)

ONL_ERR outputs a value describing possibly existing communication errors between the master coupler and the slave. ONL_ERR is only valid if DONE = TRUE and ERR = FALSE.

The error IDs of ONL_ERR correspond to the IDs of output COM_ERR.EVENT of the block DNM_STATE. They are described in the table provided in the description for this block.

GEN_ERR BYTE (general error)

GEN_ERR outputs the ID of the last error telegram of the slave in accordance with the DeviceNet specification.

GEN_ERR exclusively refers to connection errors (ONL_ERR = 35). Therefore, the error ID of GEN_ERR always has to be considered together with output ONL_ERR. GEN_ERR is only valid if DONE = TRUE, ERR = FALSE and ONL_ERR = 35.

The following table lists the possible values of DEV_ERR and their meanings.

GEN_ERR	Meaning
2	Resource not available
8	Service not supported
9	Invalid attribute value
11	Request mode already active
12	Object status conflict
14	Attribute cannot be changed
15	Privilege violation
16	Device status conflict
17	Response data too long
19	Insufficient data
20	Attribute not supported
21	Too much data
22	Object already exists

ADD_ERR BYTE (additional error)

ADD_ERR outputs additional information about the connection error reported at GEN_ERR. Therefore, ADD_ERR always has to be considered together with output GEN_ERR. ADD_ERR is only valid if DONE = TRUE, ERR = FALSE and GEN_ERR <> 0.

EXP_TOUT WORD (expected packet rate timeout)

Output EXP_TOUT indicates how often response monitoring for this slave was performed.

The DeviceNet bus parameters contain a configurable heartbeat timer. This timer determines at which time intervals the master should check the availability of the slaves. EXP_TOUT contains the number of timeouts for the requested slave. Therefore, EXP_TOUT can be used to make a statement about the quality of the connection to this slave. The value of this output is only valid if DONE = TRUE and ERR = FALSE.

Function call in IL

```
CAL DevDiag (
    EN      := DevDiag_EN,
    SLOT    := DevDiag_SLOT,
    MAC_ID  := DevDiag_MAC_ID)

LD  DevDiag.DONE
ST  DevDiag_DONE

LD  DevDiag.ERR
ST  DevDiag_ERR

LD  DevDiag.ERNO
ST  DevDiag_ERNO

LD  DevDiag.STATE_1
ST  DevDiag_STATE_1

LD  DevDiag.DEV_STATE
ST  DevDiag_DEV_STATE

LD  DevDiag.ONL_ERR
ST  DevDiag_ONL_ERR

LD  DevDiag.GEN_ERR
ST  DevDiag_GEN_ERR

LD  DevDiag.ADD_ERR
ST  DevDiag_ADD_ERR

LD  DevDiag.EXP_TOUT
ST  NDevDiag_EXP_TOUT
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DevDiag(  EN      := DevDiag_EN,
          SLOT    := DevDiag_SLOT,
          MAC_ID  := DevDiag_MAC_ID;

DevDiag_DONE      := DevDiag.DONE;
DevDiag_ERR       := DevDiag.ERR;
DevDiag_ERNO      := DevDiag.ERNO;
DevDiag_STATE_1   := DevDiag.STATE_1;
DevDiag_DEV_STATE := DevDiag.DEV_STATE;
DevDiag_ONL_ERR   := DevDiag.ONL_ERR;
DevDiag_GEN_ERR   := DevDiag.GEN_ERR;
DevDiag_ADD_ERR   := DevDiag.ADD_ERR;
DevDiag_EXP_TOUT  := DevDiag.EXP_TOUT;
```

STATE_1 DNM_DEVICESTATUS_1_TYPE

Output STAT_1 of the block DNM_DEV_DIAG displays different diagnosis bits as a structure of the type DNM_DEVICESTATUS_1_TYPE. In the DeviceNet library the structure DNM_DEVICESTATUS_1_TYPE is declared as follows:

```
TYPE DNM_DEVICESTATUS_1_TYPE:
STRUCT
    NO_RESPONSE:  BOOL;
    reserved1:    BOOL;
    PRM_FAULT:    BOOL;
    CFG_FAULT:    BOOL;
    UCMM_Support: BOOL;
    reserved2:    BOOL;
    reserved3:    BOOL;
    DEACTIVATED:  BOOL;
END_STRUCT
END_TYPE
```

NO_RESPONSE BOOL (no response)

If this bit is set, the slave with the MAC ID specified at block input MAC_ID does not respond to the master's requests. Normally NO_RESPONSE should be set to FALSE.

reserved1 BOOL

This bit is reserved and currently not in use.

PRM_FAULT BOOL (parameter fault)

This bit is set, if write access to at least one configured attribute of the slave was refused. In this case, a parameterization error occurred.

CFG_FAULT BOOL (configuration fault)

This bit is set, if the actual length of transmitted data (actual configuration) differs from the configured length (nominal configuration). In this case, a configuration error occurred.

UCMM_SUPPORT BOOL

This bit is reserved and currently not in use.

reserved2 BOOL

This bit is reserved and currently not in use.

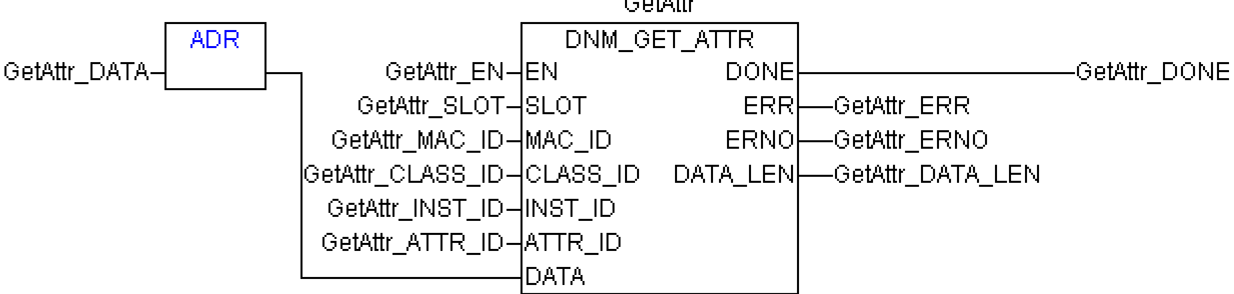
reserved3 BOOL

This bit is reserved and currently not in use.

DEACTIVATED BOOL (deactivated)

This bit is set to TRUE, if the slave defined in the configuration data of the master is deactivated and not processed.

DNM_GET_ATTR Reading an attribute from a slave object



The block DNM_GET_ATTR can be used to read individual attributes from object instances of a slave.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	DeviceNet_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DNM_GET_ATTR	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
MAC_ID	Input	BYTE	MAC ID of the relevant slave
CLASS_ID	Input	BYTE	Object class ID of the object
INST_ID	Input	BYTE	Object instance number
ATTR_ID	Input	BYTE	Attribute number within the object
DATA	Input	DWORD	Address from which on the read data are stored (via ADR operator)
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
DATA_LEN	Output	BYTE	Length of the read data in bytes

Description

The block DNM_GET_ATTR can be used to read individual attributes from slave objects.

Every time a FALSE->TRUE edge is applied to input EN, DNM_GET_ATTR reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

The DeviceNet standard describes an abstract object model (see also System technology of the DeviceNet couplers). There, different standard classes are defined which represent a kind of type definitions for the objects. In addition, further manufacturer-specific classes can be defined for a DeviceNet module. Objects are instances of the corresponding class. These objects contain attributes that can be read using the block DNM_GET_ATTR.

Some objects have to be available in every DeviceNet module by default. Normally, these standard objects are not accessed by the user program during running operation, even though this is possible in principle. Access is usually restricted to optional additional objects or their attributes. These additional slave attributes can be displayed in SYCON.net by selecting Device configuration | Parameter data. This view displays a list of available attributes for the specific slave. Each entry contains the class ID (Class ID) and the instance number (Instance ID) of the object, the number (Attribute ID) and data type of the specific attribute, the valid access types (read/write) and an attribute description as well as the permitted limit values. This information can be used to select the desired CLASS_ID, INST_ID and ATTR_ID when using a function block of the type DNM_GET_ATTR. Furthermore, it can be used to determine the variable type required at block input DATA (via ADR operator) in order to be able to store and interpret the received data.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at the outputs ERR and ERNO and the termination of the request processing is indicated by DONE = TRUE.

During the processing of a request, state changes at input EN are recognized but not evaluated.

Since block execution requires bus access, the data are available in the next cycle after activating the block at the earliest.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

MAC_ID BYTE (medium access control identifier)

Input MAC_ID is used to specify the MAC ID of the slave from which an attribute is to be requested.

CLASS_ID BYTE (class identifier)

Input CLASS_ID is used to specify the class ID of the object containing the attribute to be read (compare to Class entry in Parameter data list in SYCON.net).

INST_ID BYTE (instance identifier)

Input INST_ID is used to specify the instance number of the object containing the attribute to be read (compare to Inst. entry in Parameter data list in SYCON.net).

ATTR_ID BYTE (attribute identifier)

Input ATTR_ID is used to specify the number of the attribute to be read within the object containing the attribute (compare to Attr. entry in Parameter data list in SYCON.net).

DATA DWORD (data)

At input DATA, the address of the variable to which the received attribute data are to be written is specified via the ADR operator (for the data format refer to the Type entry in the Parameter data list in SYCON.net). The received data are only valid if DONE = TRUE, ERR = FALSE and DATA_LEN > 0.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

DATA_LEN BYTE (data length)

After the process has been completed successfully, output at DATA_LEN displays the length of the received attribute data in bytes. Since block execution requires bus access, the data are available in the next cycle after activating the block at the earliest. The value of this output is only valid if DONE = TRUE and ERR = FALSE.

Function call in IL

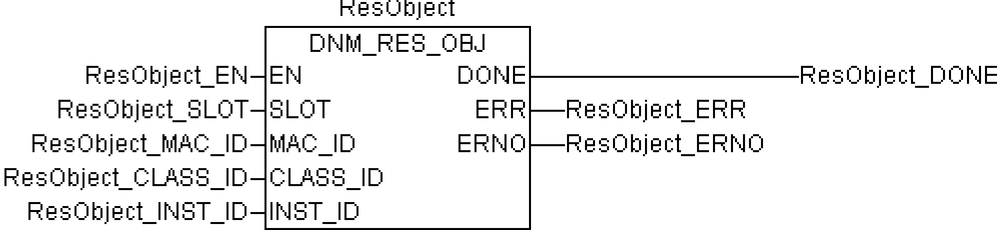
```
LD   GetAttr_DATA
ADR
ST   ADR_GetAttr_DATA

CAL GetAttr (
    EN       := GetAttr_EN,
    SLOT     := GetAttr_SLOT,
    MAC_ID   := GetAttr_MAC_ID,
    CLASS_ID := GetAttr_CLASS_ID,
    INST_ID  := GetAttr_INST_ID,
    ATTR_ID  := GetAttr_ATTR_ID,
    DATA     := ADR_GetAttr_DATA)

LD   GetAttr.DONE
ST   GetAttr_DONE

LD   GetAttr.ERR
ST   GetAttr_ERR

LD   GetAttr.ERNO
ST   GetAttr_ERNO

LD   GetAttr.DATA_LEN
ST   GetAttr_DATA_LEN
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
GetAttr(  EN        := GetAttr_EN,
          SLOT      := GetAttr_SLOT,
          MAC_ID    := GetAttr_MAC_ID,
          CLASS_ID  := GetAttr_CLASS_ID,
          INST_ID   := GetAttr_INST_ID,
          ATTR_ID   := GetAttr_ATTR_ID,
          DATA      := ADR(GetAttr_DATA));

GetAttr_DONE        := GetAttr.DONE;
GetAttr_ERR         := GetAttr.ERR;
GetAttr_ERNO        := GetAttr.ERNO;
GetAttr_DATA_LEN    := GetAttr.DATA_LEN;
```

DNM_RES_OBJ Resetting a slave object



The block DNM_RES_OBJ can be used to reset a slave object.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	DeviceNet_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DNM_RES_OBJ	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
MAC_ID	Input	BYTE	MAC ID of the relevant slave
CLASS_ID	Input	BYTE	Object class ID of the object
INST_ID	Input	BYTE	Object instance number
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block DNM_RES_OBJ can be used to reset an instance of a class (object) in a slave.

Every time a FALSE->TRUE edge is applied to input EN, DNM_RES_OBJ reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

The DeviceNet standard describes an abstract object model (see also System technology of the DeviceNet couplers). There, different standard classes are defined which represent a kind of type definitions for the objects. In addition, further manufacturer-specific classes can be defined for a DeviceNet module. Objects are instances of the corresponding class. These objects can be reset using the block DNM_RES_OBJ.

Some objects have to be available in every DeviceNet module by default. Normally, these standard objects are not accessed by the user program during running operation, even though this is possible in principle. Access is usually restricted to optional additional objects or their attributes.

These additional slave attributes can be displayed in SYCON.net by selecting Device configuration | Parameter data. This view displays a list of available attributes for the specific slave. Each entry contains the class ID (Class ID) and the instance number (Instance ID) of the object, the number (Attribute ID) and data type of the specific attribute, the valid access types (read/write) and an attribute description as well as the permitted limit values. This information can be used to reset the desired CLASS_ID and INST_ID when using a function block of the type DNM_RES_OBJ.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at the outputs ERR and ERNO and the termination of the request processing is indicated by DONE = TRUE.

During the processing of a request, state changes at input EN are recognized but not evaluated.

Since block execution requires bus access, the data are available in the next cycle after activating the block at the earliest.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

MAC_ID BYTE (medium access control identifier)

Input MAC_ID is used to specify the MAC ID of the slave an object of which is to be reset.

CLASS_ID BYTE (class identifier)

Input CLASS_ID is used to specify the class ID of the object to be reset (compare to Class entry in Parameter data list in SYCON.net).

INST_ID BYTE (instance identifier)

Input INST_ID is used to specify the instance number of the object to be reset (compare to Inst. entry in Parameter data list in SYCON.net).

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

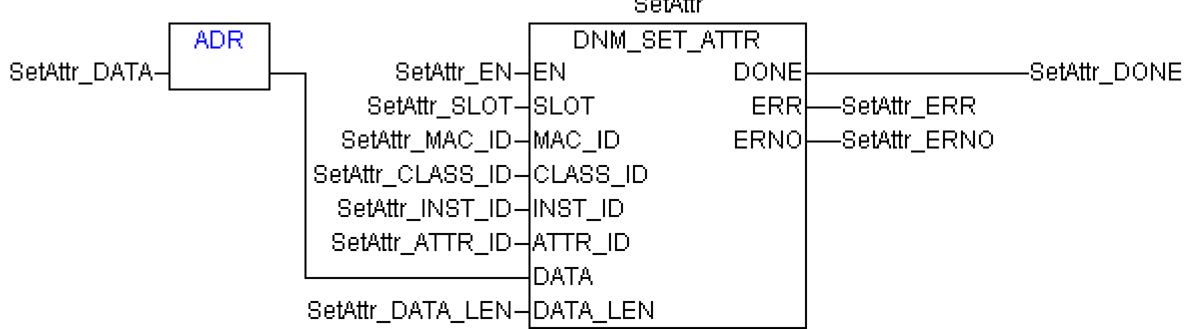
```
CAL ResObject (
    EN       := ResObject_EN,
    SLOT     := ResObject_SLOT,
    MAC_ID   := ResObject_MAC_ID,
    CLASS_ID := ResObject_CLASS_ID,
    INST_ID  := ResObject_INST_ID)

LD  ResObject.DONE
ST  ResObject_DONE

LD  ResObject.ERR
ST  ResObject_ERR

LD  ResObject.ERNO
ST  ResObject_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
ResObject( EN       := ResObject_EN,
           SLOT     := ResObject_SLOT,
           MAC_ID   := ResObject_MAC_ID,
           CLASS_ID := ResObject_CLASS_ID,
           INST_ID  := ResObject_INST_ID);

ResObject_DONE      := ResObject.DONE;
ResObject_ERR       := ResObject.ERR;
ResObject_ERNO      := ResObject.ERNO;
```

DNM_SET_ATTR Writing an attribute to a slave object



The block DNM_SET_ATTR can be used to write individual attributes of object instances in a slave.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	DeviceNet_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DNM_SET_ATTR	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
MAC_ID	Input	BYTE	MAC ID of the relevant slave
CLASS_ID	Input	BYTE	Object class ID of the object
INST_ID	Input	BYTE	Object instance number
ATTR_ID	Input	BYTE	Attribute number within the object
DATA	Input	DWORD	Address from which on the data to be written are stored (via ADR operator)
DATA_LEN	Input	BYTE	Length of the data to be written (in bytes)
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block DNM_SET_ATTR can be used to write individual attributes in slave objects.

Every time a FALSE->TRUE edge is applied to input EN, DNM_SET_ATTR reads the data at its inputs and sends a corresponding request message to the coupler. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

The DeviceNet standard describes an abstract object model (see also System technology of the DeviceNet couplers). There, different standard classes are defined which represent a kind of type definitions for the objects. In addition, further manufacturer-specific classes can be defined for a DeviceNet module. Objects are instances of the corresponding class. These objects contain attributes that can be written using the block DNM_SET_ATTR.

Some objects have to be available in every DeviceNet module by default. Normally, these standard objects are not accessed by the user program during running operation, even though this is possible in principle. Access is usually restricted to optional additional objects or their attributes. These additional slave attributes can be displayed in SYCON.net by selecting Device configuration | Parameter data. This view displays a list of available attributes for the specific slave. Each entry contains the class ID (Class ID) and the instance number (Instance ID) of the object, the number (Attribute ID) and data type of the specific attribute, the valid access types (read/write) and an attribute description as well as the permitted limit values. This information can be used to select the desired CLASS_ID, INST_ID and ATTR_ID when using a function block of the type DNM_GET_ATTR. Furthermore, it can be used to determine the variable type required at block input DATA (via ADR operator) and the value to be assigned to this variable in order to obtain the required result.

EN BOOL (enable)

If a FALSE->TRUE edge is applied to input EN, all further inputs are read. If the input values are valid, a corresponding request message is sent to the coupler. While this request message is processed, output DONE is set to FALSE. If at least one input value is invalid, an error is indicated at the outputs ERR and ERNO and the termination of the request processing is indicated by DONE = TRUE.

During the processing of a request, state changes at input EN are recognized but not evaluated.

Since block execution requires bus access, the data are available in the next cycle after activating the block at the earliest.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

MAC_ID BYTE (medium access control identifier)

Input MAC_ID is used to specify the MAC ID of the slave in which an attribute is to be written.

CLASS_ID BYTE (class identifier)

Input CLASS_ID is used to specify the class ID of the object containing the attribute to be written (compare to Class entry in Parameter data list in SYCON.net).

INST_ID BYTE (instance identifier)

Input INST_ID is used to specify the instance number of the object containing the attribute to be written (compare to Inst. entry in Parameter data list in SYCON.net).

ATTR_ID BYTE (attribute identifier)

Input ATTR_ID is used to specify the number of the attribute to be written within the object containing the attribute (compare to Attr. entry in Parameter data list in SYCON.net).

DATA DWORD (data)

At input DATA, the address of the variable containing the attribute data to be transmitted is specified via the ADR operator (for the data format refer to the Type entry in the Parameter data list in SYCON.net).

DATA_LEN BYTE (data length)

At input DATA_LEN, the length of the data to be transmitted which are stored in the variable at address DATA is specified as a byte value.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

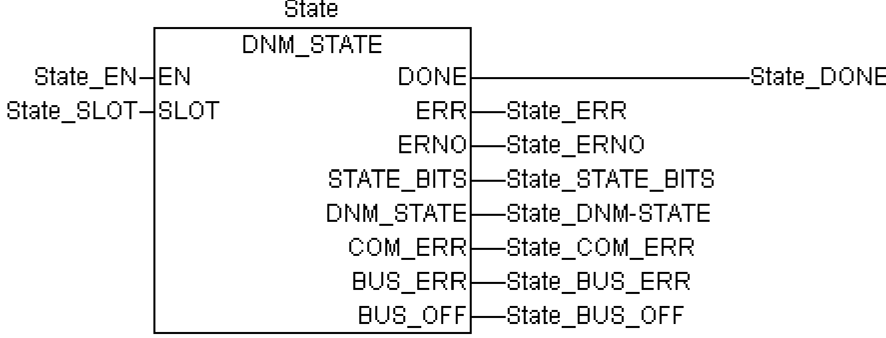
```
LD  SetAttr_DATA
ADR
ST  ADR_SetAttr_DATA

CAL SetAttr (
    EN       := SetAttr_EN,
    SLOT     := SetAttr_SLOT,
    MAC_ID   := SetAttr_MAC_ID,
    CLASS_ID := SetAttr_CLASS_ID,
    INST_ID  := SetAttr_INST_ID,
    ATTR_ID  := SetAttr_ATTR_ID,
    DATA     := ADR_SetAttr_DATA,
    DATA_LEN := SetAttr_DATA_LEN)

LD  SetAttr.DONE
ST  SetAttr_DONE

LD  SetAttr.ERR
ST  SetAttr_ERR

LD  SetAttr.ERNO
ST  SetAttr_ERNO

LD  SetAttr.DATA_LEN
ST  SetAttr_DATA_LEN
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
SetAttr(  EN        := SetAttr_EN,
          SLOT      := SetAttr_SLOT,
          MAC_ID    := SetAttr_MAC_ID,
          CLASS_ID  := SetAttr_CLASS_ID,
          INST_ID   := SetAttr_INST_ID,
          ATTR_ID   := SetAttr_ATTR_ID,
          DATA      := ADR(SetAttr_DATA),
          DATA_LEN  := SetAttr_DATA_LEN);

SetAttr_DONE        := SetAttr.DONE;
SetAttr_ERR         := SetAttr.ERR;
SetAttr_ERNO        := SetAttr.ERNO;
```

DNM_STATE Reading the DeviceNet coupler status



DNM_STATE outputs the status of a DeviceNet coupler. The outputs provide information about the communication status and error events.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	DeviceNet_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DNM_STATE	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
STATE_BITS	Output	DNM_STATE_BITS_TYPE	Atypical communication states
DNM_STATE	Output	BYTE	General state of the DeviceNet master
COM_ERR	Output	DNM_COM_ERR_TYPE	Communication error
BUS_ERR	Output	WORD	Number of bus failures
BUS_OFF	Output	WORD	Number of bus outages

Description

The block DNM_STATE outputs the current status of the DeviceNet coupler.

DNM_STATE is active if input EN = TRUE. If the block is active and if no errors occurred during block processing, the current values are permanently displayed at the outputs.

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

STATE_BITS DNM_STATE_BITS_TYPE (state bits)

Output STATE_BITS indicates atypical communication states of the DeviceNet coupler. STATE_BITS is only valid if EN = TRUE and ERR = FALSE.

The structure of the type DNM_STATE_BITS_TYPE is defined in the DeviceNet library (see description below).

DNM_STATE BYTE (devicenet master state)

DNM_STATE outputs the general communication state of the DeviceNet master. The following states are defined:

State		Meaning
Dec	Hex	
0	00	OFFLINE
64	40	STOP
128	80	CLEAR
192	C0	OPERATE

DNM_STATE = OFFLINE

If DNM_STATE is set to OFFLINE, the DeviceNet coupler performs an initialization. After the initialization phase is completed, the coupler changes to STOP state.

DNM_STATE = STOP

The coupler is completely initialized, if DNM_STATE has the value STOP. In this state the coupler is ready to receive configuration data. There is no data exchange with the slaves. The coupler has this state if no user program is running.

DNM_STATE = CLEAR

If the user program is started, the coupler changes from STOP to CLEAR and starts to establish the connections defined during configuration. After the setup is completed successfully, the coupler moves to OPERATE state. If an error occurs during parameterization, the coupler changes back to STOP state.

DNM_STATE = OPERATE

Normally, the coupler is in OPERATE state when a user program is running. In this state the master exchanges I/O data with the slaves. If an error occurs during this process and if 'Auto Clear Mode' has been selected during configuration, the coupler changes back to CLEAR state and tries to establish the connections again. If 'Auto Clear Mode' has not been selected during configuration, the coupler remains in OPERATE state in case of an error. If the user program is stopped, the coupler also changes back to STOP state.

DNM_STATE is only valid if EN = TRUE and ERR = FALSE.

COM_ERR DNM_COM_ERR_TYPE (communication error)

Output COM_ERR indicates possibly occurring communication errors. COM_ERR is only valid if EN = TRUE and ERR = FALSE.

The structure of the type DNM_COM_ERR_TYPE is defined in the DeviceNet library and described below together with the possible errors.

BUS_ERR WORD (bus error)

BUS_ERR outputs the number of occurred bus failures. A bus failure occurs if the internal error frame counter exceeds a specific value. BUS_ERR is only valid if EN = TRUE and ERR = FALSE.

BUS_OFF WORD (bus off)

BUS_OFF outputs how often the coupler has been excluded from bus activities. An exclusion from the bus activities is performed in case of an overflow of the internal error frame counter. The coupler is automatically re-initialized after each overflow. BUS_OFF is only valid, if EN = TRUE and ERR = FALSE.

Function call in IL

```
CAL   State (
      EN       := State_EN,
      SLOT     := State_SLOT)

LD    State.DONE
ST    State_DONE

LD    State.ERR
ST    State_ERR

LD    State.ERNO
ST    State_ERNO

LD    State.STATE_BITS
ST    State_STATE_BITS

LD    State.DNM_STATE
ST    State_DNM_STATE

LD    State.COM_ERR
ST    State_COM_ERR

LD    State.BUS_ERR
ST    State_BUS_ERR
```

```
LD    State.BUS_OFF
ST    State_BUS_OFF
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
State ( EN          := State_EN,
        SLOT        := State_SLOT);

State_DONE          := State.DONE;
State_ERR           := State.ERR;
State_ERNO          := State.ERNO;
State_STATE_BITS    := State.STATE_BITS;
State_DNM_STATE     := State.DNM_STATE;
State_COM_ERR       := State.COM_ERR;
State_BUS_ERR       := State.BUS_ERR;
State_BUS_OFF       := State.BUS_OFF;
```

STATE_BITS   DNM_STATE_BITS_TYPE

The structure STATE_BITS consists of six boolean variables displaying different communication states. In the DeviceNet library the data type DNM_STATE_BITS_TYPE is declared as follows:

```
TYPE DNM_STATE_BITS_TYPE:
STRUCT
   CTRL:        BOOL;
   AUTO_CLR:    BOOL;
   NO_EXCH:     BOOL;
   FATAL:       BOOL;
   reserved1:   BOOL;
   reserved2:   BOOL;
   DUP_MAC_CHK: BOOL;
   DUP_MAC_DET: BOOL;
END_STRUCT
END_TYPE
```

CTRL   BOOL   (control)

If this bit is TRUE, a parameter setting error occurred. During normal operation, CTRL should be FALSE. If this is not the case, the parameter and configuration data have to be checked.

AUTO_CLR   BOOL   (auto clear)

If AUTO_CLR is set to TRUE, the coupler has stopped data exchange with all slaves due to communication errors and has changed back to CLEAR state (see DNM_STATE).

NO_EXCH   BOOL   (no exchange)

This bit is set to TRUE, if no exchange of process data can be performed with one or several slaves. The error can be caused by the configuration data or the slaves themselves.

FATAL   BOOL   (fatal)

If FATAL is set to TRUE, no communication via DeviceNet is possible due to a fatal bus error (e.g. bus line short circuit). In this case, all bus lines have to be checked.

reserved1   BOOL

This bit is reserved and currently not in use.

reserved2 BOOL

This bit is reserved and currently not in use.

DUP_MAC_CHK BOOL (duplicate MAC-ID check)

If DUP_MAC_CHK = TRUE, the coupler is occupied with checking the bus for duplicate MAC IDs. The check is completed, if the coupler is able to find other DeviceNet modules on the bus to cross-check the MAC ID for duplicates and if it did not detect any duplicate MAC IDs during this check. The completion of the check is indicated by DUP_MAC_CHK = FALSE.

DUP_MAC_DET BOOL (duplicate MAC-ID detect)

If DUP_MAC_DET = TRUE, the coupler has detected another DeviceNet module with the same MAC ID on the bus. In this case, the MAC IDs of all devices on the bus have to be checked.

COM_ERR DNM_COM_ERR_TYPE

Communication errors can be located more detailed using COM_ERR. The output COM_ERR is represented as a structure of the type DNM_COM_ERR_TYPE. In the DeviceNet library this data type is declared as follows:

```
TYPE DNM_COM_ERR_TYPE:
STRUCT
   ADDRESS: BYTE;
   EVENT:   BYTE;
END_STRUCT
END_TYPE
```

ADDRESS BYTE (address)

If an error occurs, ADDRESS contains the bus address of the faulty device (0 to 63). If ADDRESS has the value 255, the error is located in the coupler itself.

EVENT BYTE (event)

In case of an error, EVENT contains the error causing event. The event either refers to single node addresses (ADDRESS 0..63) or to the coupler itself (ADDRESS = 255).

ADDRESS <> 255 Error at subscribers with node address "ADDRESS"

Event	Meaning	Error source	Cause / Remedy
1	Slave monitoring failed after slave was already operational	Slave	Check that the slave is switched on and running
30	Slave access timeout	Slave	Slave is not responding; check baud rate and MAC ID of the slave
32	Slave refuses access with error ID	Slave	Get slave diagnosis to evaluate the error ID
35	Slave responds with a connection error during allocation phase	Slave	Get slave diagnosis to evaluate the extended error ID
36	Actual length of produced connection (input data from the master's point of view) differs from the configured length	Slave / Configuration	Get slave diagnosis to determine the actual length
37	Actual length of consumed connection (output data from the master's point of view) differs from the configured length	Slave / Configuration	Get slave diagnosis to determine the actual length
38	Unknown and unprocessed slave	Slave /	Get slave diagnosis to determine

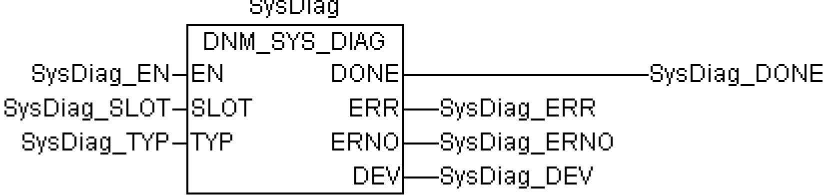
	response to service telegram	Coupler	the actual length
39	Connection already requested	Slave	Connection is terminated automatically
40	Number of CAN message data bytes is not 4 when reading the length of the produced or the consumed connection	Slave	Slave behavior does not conform to the standard, communication with slave not possible
41	Predefined master-slave connection already exists	Slave / Coupler	Connection is terminated automatically
42	Slave polling response length differs from the produced connection length	Slave	-
43	Sequence error in slave polling response	Slave	The first two segments in multiplex transfer were received
44	Fragmentation error in slave polling response	Slave	Fragmentation counter differs from the expected value in multiplex transfer
45	Sequence error in slave polling response	Slave	Middle or last segment was received prior to first segment
46	Bit strobe response length differs from the produced connection length	Slave	-
47	Sequence error in state change response or cyclical response of the slave	Slave	The first two segments in multiplex transfer were received
48	Fragmentation error in state change response or cyclical response of the slave	Slave	Fragmentation counter differs from the expected value in multiplex transfer
49	Sequence error in state change response or cyclical response of the slave	Slave	Middle or last segment was received prior to first segment
51	Length of state change response or cyclical response of the slave differs from the produced connection length	Slave	-
52	UCMM group not supported	Slave	Change UCMM group
59	Actual configuration data contain duplicate device addresses	Configuration	Each device must have a unique MAC ID; edit configuration data
60	Invalid length indicator of a slave data set	Configuration	Error when loading the configuration data; contact customer support
61	Additional table of predefined master-slave connections has invalid length indicator	Configuration	Error when loading the configuration data; contact customer support
62	Invalid length indicator of the master-slave I/O configuration table	Configuration	Error when loading the configuration data; contact customer support
63	Predefined master-slave I/O configuration differs from the additional table	Configuration	Different number of I/O modules and configured offset addresses; contact customer support
64	Invalid length indicator of the parameter data set	Configuration	Value of length indicator too low; contact customer support
65	Number of inputs declared in the additional table differs from the number of inputs defined in the I/O configuration	Configuration	Each I/O configuration entry must have exactly one entry assigned in the additional table; contact customer support
66	Number of outputs declared in the additional table differs from the number of outputs defined in the I/O configuration	Configuration	Each I/O configuration entry must have exactly one entry assigned in the additional table; contact customer support

67	I/O configuration contains unknown data type	Configuration	Only the data types BOOL, BYTE, WORD, DWORD and STRING are supported
68	Data type of a configured I/O module in a connection differs from the defined data length	Configuration	The following data types and lengths are valid: BOOL = 1 byte UINT8 = BYTE = 1 byte UINT16 = WORD = 2 bytes UINT32 = DWORD = 4 bytes
69	The configured output addresses of a module (offset + length) exceed the maximum value of 3584	Configuration	The process data image is limited to 3584 bytes
70	The configured input addresses of a module (offset + length) exceed the maximum value of 3584	Configuration	The process data image is limited to 3584 bytes
71	Unknown predefined connection type	Configuration	Only the connection types cyclic, poll, state change and bit strobe are supported
72	Multiple parallel connections defined	Configuration	Only one connection type is supported per slave
73	The configured expected packet rate of a connection is smaller than the production inhibit time	Configuration	Expected packet rate has to be greater than the production inhibit time

ADDRESS = 255 Coupler error

Event	Meaning	Error source	Cause / Remedy
52	Unknown process data handshake mode	Configuration	Contact customer support
53	Invalid data transfer rate	Configuration	Check the data transfer rate; contact customer support
54	Invalid MAC ID	Configuration	Check the MAC ID specified in configuration data
57	Duplicate MAC ID found on the bus	Configuration / other device	Check the MAC ID of all devices specified in configuration data
58	No device entry in current configuration data	Configuration	Error when loading the configuration data; contact customer support
210	No configuration data	Configuration / Coupler	Load configuration data into coupler
212	Error while reading database	Configuration / Coupler	Contact customer support
220	Watchdog error	Controller	Contact customer support
221	No process data handshake	Controller	Contact customer support

DNM_SYS_DIAG Displaying status surveys of all slaves



The block DNM_SYS_DIAG outputs a bit field as a status survey of all slaves (clients) at output DEV. Three different survey types can be selected via input TYP.

Block data

Available as of PLC runtime system:	V1.1	Remark:
Included in library:	DeviceNet_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DNM_SYS_DIAG	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot (module number) of the coupler
TYP	Input	BYTE	Selection of the survey type
DONE	Output	BOOL	Data package available or error occurred
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
DEV	Output	ARRAY[0...63] OF BOOL	Status survey of the slaves

Description

The block CANOM_SYS_DIAG outputs different types of status surveys of all slaves. Three survey types can be selected:

- configuration survey
- I/O data exchange survey
- diagnosis survey

EN BOOL (enable)

The block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the block is active, the current values are available at the outputs.

SLOT BYTE (slot)

At input SLOT, the coupler slot (module number) to be used by the block is selected.

The internal coupler always has the module number 0. All external couplers are serially numbered from right to left, starting with module number 1.

TYP BYTE (type)

Input TYP is used to select the type of status survey to be displayed at output DEV.

TYP = 1 Configuration survey

Output DEV displays all slaves that are successfully connected to the master. Please note that the master only establishes connections to slaves which have been assigned to the master during the definition of the configuration data.

TYP = 2 Data exchange survey

Output DEV displays the slaves with which the master is exchanging I/O data. Data exchange can only be performed with slaves that have been configured by the master itself. The data exchange survey can only be requested, if the coupler is in OPERATE state.

TYP = 3 Diagnosis survey

Output DEV indicates which slaves report a diagnosis. The diagnosis survey can only be requested if the coupler is in OPERATE state.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion of the request or after abortion of the block processing due to an error, DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during block processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

DEV ARRAY[0...63] OF BOOL (device)

Output DEV outputs the status survey as a bit field. Each individual bit within this field represents one slave. The index number corresponds to the slave's bus address (MAC ID). If a bit is set to TRUE, the state selected at input TYP applies to the corresponding slave.

If e.g. TYP = 1 is selected and DEV[2] = TRUE, the slave with the MAC ID 2 was successfully configured by the master.

If DEV[2] = FALSE, the configuration of this specific slave has not yet been completed or the slave is not part of the master's configuration data.

If TYP = 2 and DEV[2] = TRUE, the master is exchanging I/O data with the slave having the MAC ID 2. However, if DEV[2] = FALSE, the master is currently not exchanging I/O data with the slave. The master can only exchange data with slaves the master itself previously brought into operation.

When TYP = 3, for example DEV[2] = TRUE means that the slave with the MAC ID 2 reports a diagnosis. Then, a detailed diagnosis can be requested using the block DNM_DEV_DIAG.

The bit field output at DEV is only valid if EN = TRUE and ERR = FALSE.

Function call in IL

```
CAL SysDiag (
    EN       := SysDiag_EN,
    SLOT     := SysDiag_SLOT,
    TYP      := SysDiag_TYP)

LD  SysDiag.DONE
ST  SysDiag_DONE

LD  SysDiag.ERR
ST  SysDiag_ERR

LD  SysDiag.ERNO
ST  SysDiag_ERNO

LD  SysDiag.DEV
ST  SysDiag_DEV
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
SysDiag( EN   := SysDiag_EN,
         SLOT := SysDiag_SLOT,
         TYP  := SysDiag_TYP);

SysDiag_DONE  := SysDiag.DONE;
SysDiag_ERR   := SysDiag.ERR;
SysDiag_ERNO  := SysDiag.ERNO;
SysDiag_DEV   := SysDiag.DEV;
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

Components of the DeviceNet library 2

D

DNM_DEV_DIAG Polling diagnosis data from a slave 4

DNM_GET_ATTR Reading an attribute from a slave object 10

DNM_RES_OBJ Resetting a slave object 14

DNM_SET_ATTR Writing an attribute to a slave object 17

DNM_STATE Reading the DeviceNet coupler status 21

DNM_SYS_DIAG Displaying status surveys of all slaves 28

G

Glossary 31

O

Overview of blocks arranged according to their call names 3

S

Special characteristics of the DeviceNet library 2

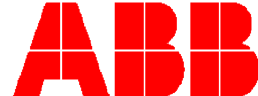
Software Description

AC500

Scalable PLC
for Individual Automation

DC541
Function Block Library

DC541



Contents

DC541 Library 2

Preconditions for the use of the library 2

Special characteristics of the DC541 library 2

Components of the DC541 library 2

Overview of blocks arranged according to their call names 3

Integrated visualization of blocks contained in the DC541 library 3

DC541_32BIT_CNT 32 bit encoder 5

DC541_FREQ Time and frequency measurement 13

DC541_FREQ_FAST Fast time and frequency measurement 20

DC541_FREQ_OUT Frequency output 26

DC541_FWD_CNT 32 bit count-up counter 31

DC541_GET_CFG Output of DC541 configuration details 37

DC541_INT_IN Display of the interrupt initiating source 42

DC541_IO Reading/writing the inputs and outputs of the DC541 45

DC541_LIMIT Limit value monitoring for the 32 bit counter 48

DC541_PWM Pulse-width modulator 53

DC541_STATE Status polling for the DC541 57

Glossary 60

Index 62

DC541 Library

Preconditions for the use of the library

Note:

The blocks contained in the DC541 library can only be executed in RUN mode of the PLC, but not in simulation mode.

The function blocks of the library DC541_AC500_V11.lib are available in AC500 control systems with a runtime system of version V1.1.2 or later.

Special characteristics of the DC541 library

The DC541 library contains all blocks necessary for using the DC541. The configuration of the DC541 is done in the PLC configuration. A detailed description of the AC500 PLC configuration can be found in the chapter "System Technology of the CPUs / PLC configuration".

Once a DC541 has been added to the PLC configuration, the library DC541_AC500_V11.lib is automatically included with the next compilation of the project.

Components of the DC541 library

The DC541 library contains the following function blocks:

Group: Diagnosis	
DC541_GET_CFG	Output of DC541 configuration details
DC541_STATE	Status polling for the DC541

Group: IO	
DC541_INT_IN	Display of the interrupt initiating source
DC541_IO	Reading/writing the inputs and outputs of the DC541

Group: Counters	
DC541_32BIT_CNT	32 bit encoder
DC541_FREQ	Time and frequency measurement
DC541_FREQ_FAST	Fast time and frequency measurement
DC541_FREQ_OUT	Frequency output
DC541_FWD_CNT	32 bit count-up counter
DC541_LIMIT	Limit value monitoring for the 32 bit counter
DC541_PWM	Pulse-width modulation

Overview of blocks arranged according to their call names

Character description:

FBhV … Function block with historical values

FBnohV … Function block without historical values

F … Function

VE name	Type	Function
DC541_32BIT_CNT	FBhv	32 bit encoder
DC541_FREQ	FBhv	Time and frequency measurement
DC541_FREQ_FAST	FBhv	Fast time and frequency measurement
DC541_FREQ_OUT	FBhv	Frequency output
DC541_FWD_CNT	FBhv	32 bit count-up counter
DC541_GET_CFG	FBhv	Output of DC541 configuration details
DC541_INT_IN	FBhv	Display of the interrupt initiating source
DC541_IO	FBhv	Reading/writing the inputs and outputs of the DC541
DC541_LIMIT	FBhv	Limit value monitoring for the 32 bit counter
DC541_PWM	FBhv	Pulse-width modulation
DC541_STATE	FBhv	Status polling for the DC541

Integrated visualization of blocks contained in the DC541 library

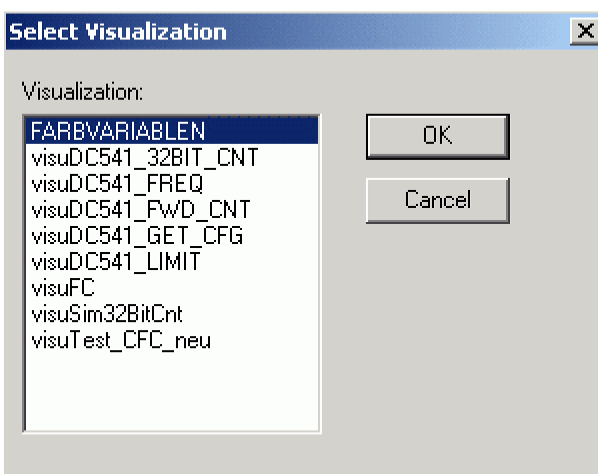
The following blocks of the library DC541_AC500_V11.LIB have an integrated visualization:

Block	Visualization name
DC541_32BIT_CN	visuDC541_32BIT_CNT
DC541_FREQ	visuDC541_FREQ
DC541_FREQ_FAST	visuDC541_FREQ_FAST
DC541_FREQ_OUT	visuDC541_FREQ_OUT
DC541_FWD_CNT	visuDC541_FWD_CNT
DC541_GET_CFG	visuDC541_GET_CFG
DC541_LIMIT	visuDC541_LIMIT

The visualization can be used to display the block outputs. If the input EN_VISU of a block is TRUE, it is also possible to control the inputs from the visualization. In order to allow the control of block inputs from the program as well as from the visualization, these inputs are declared as VAR_IN_OUT and consequently have to be provided with variables. These inputs must not be provided with direct constants.

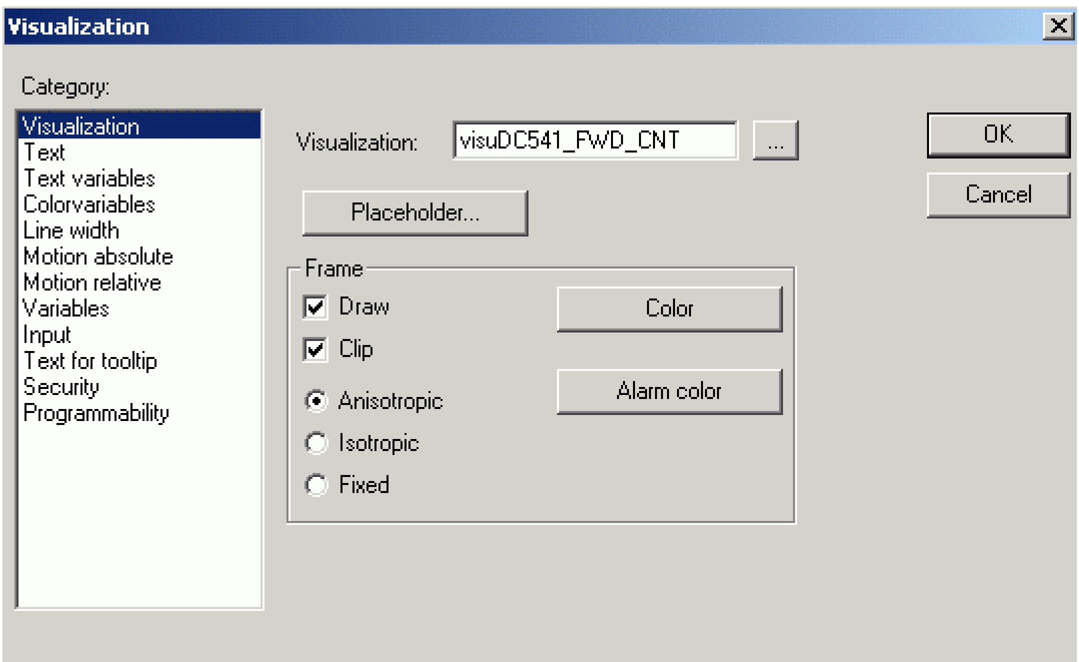
Proceed as follows to integrate the visualization into a project:

- Create a new visualization using Visualizations / Add Object (e.g. visuTestDC541).
- Insert a visualization using .
- In the appearing dialog, select the corresponding visualization for the block:



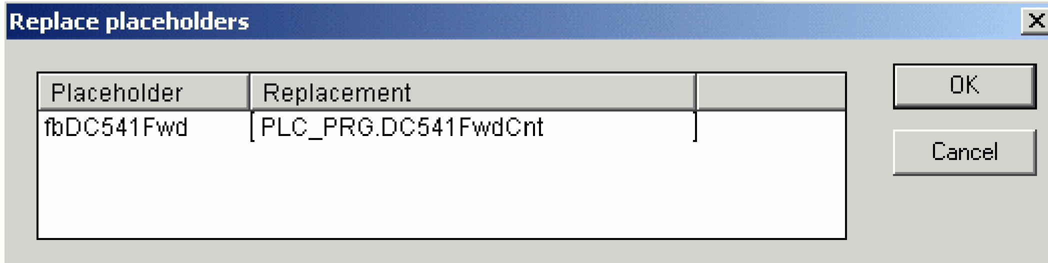
- Then, the visualization integrated just now has to be configured. Highlight the visualization with a left mouse click. Then click the right mouse button and select the function "Configure..." from the context menu.

- The configuration of the visualization is done in the appearing dialog. It is recommended to set the frame to "Fixed" in order to maintain the original width-to-height ratio and font size.



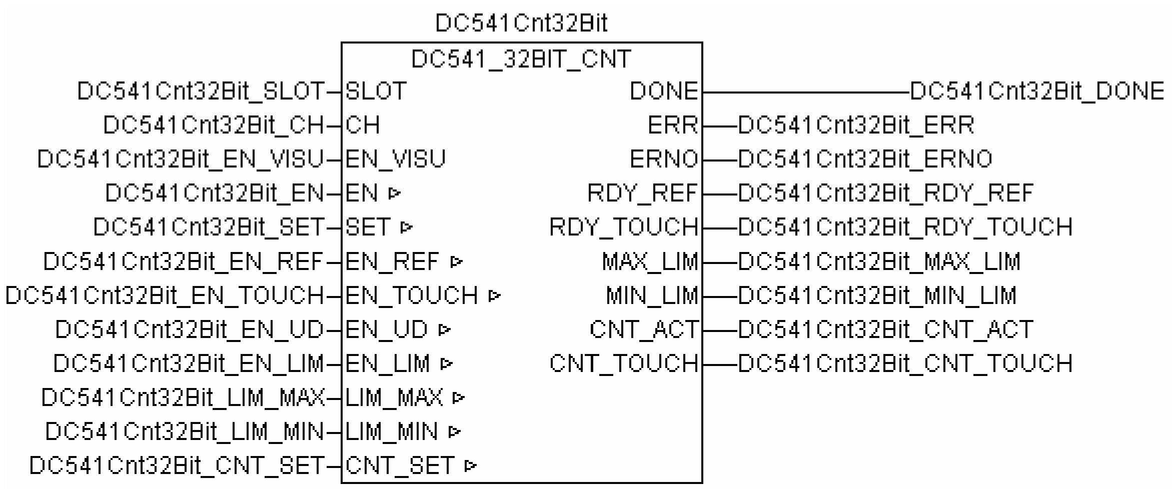
- Now the visualization has to be linked to the block instance. This is done in the dialog of the button "Placeholder".

- In the "Replacement" column of this dialog, the block instance can either be entered directly or selected using the input aid <F2>.



- By clicking on <OK> the dialogs are closed. After this, the inserted visualization has to be adapted to the correct size.

DC541_32BIT_CNT 32 bit encoder



The block DC541_32BIT_CNT is a 32 bit counter with a directional discriminator, suitable for connecting an incremental transmitter.

Block data

Available as of PLC runtime system:	V1.1.3	Remark:
Available as of DC541 firmware:	V1.1	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_32BIT_CNT	Instance name
SLOT	Input	BYTE	Slot of the DC541
CH	Input	BYTE	=0, channel number, currently channel C0 only
EN_VISU	Input	BOOL	Enabling of control via the integrated visualization of block visuDC541_32BIT_CNT
EN	Input/Output	BOOL	Enabling of the block processing
SET	Input/Output	BOOL	Set input
EN_REF	Input/Output	BOOL	Enabling of the reference point approach
EN_TOUCH	Input/Output	BOOL	Enabling of the touch trigger measurement
EN_UD	Input/Output	BOOL	Selection of counting mode: =FALSE: Incremental transmitter (tracks A/B) =TRUE: C1 up / C0 down
EN_LIM	Input/Output	BOOL	Counting mode selection: Infinite counter or limiting counter
LIM_MAX	Input/Output	DWORD	Counter end value at EN_LIM = TRUE
LIM_MIN	Input/Output	DWORD	Counter start value at EN_LIM = TRUE
CNT_SET	Input/Output	DWORD	Counter set value

DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
RDY_REF	Output	BOOL	Ready message of the reference point approach
RDY_TOUCH	Output	BOOL	Ready message touch trigger
MAX_LIM	Output	BOOL	=TRUE: LIM_MAX reached (infinite counter)
MIN_LIM	Output	BOOL	=TRUE: LIM_MIN reached (infinite counter)
CNT_ACT	Output	DWORD	Present counter value (counter content)
CNT_TOUCH	Output	DWORD	Touch trigger value

Description

This 32 bit counter is a count-up/count-down counter with a directional discriminator. The counter can be used in two counting modes.

- EN_UD=FALSE: Connection of an incremental transmitter (track A / track B, offset by 90°)
  Counting of signals up to approx. 60 kHz is possible. This corresponds to a motor with a rotational speed of 3600 rpm and a transmitter with 1000 pulses per rotation. Pulse multiplication (x2 or x4) is not used.

- EN_UD=TRUE: Up / down counter
  Counting of signals with approx. 60 kHz is possible. Count-up for signals on channel C1, count-down for signals on channel C0.

The counter always uses the channels C0...C3 of the DC541:

- C0: Track A of the incremental transmitter
- C1: Track B of the incremental transmitter
- C2 and C3: Reference cam or touch trigger

The counter can be used in two operating modes:

- Infinite counter (endless mode)
- Limiting counter (limit mode)

The operating mode is selected using input EN_LIM.

If EN_LIM = FALSE, the counter operates as an infinite counter (endless mode). An overflow occurs corresponding to the 32 bit value at 16#FFFFFFFF = 4 294 967 295. In this mode, any exceeding of the limit value LIM_MAX or falling below the limit value LIM_MIN is displayed at the outputs MAX_LIM or MIN_LIM.

If EN_LIM = TRUE (limit mode), the counting range is between the limit values LIM_MIN and LIM_MAX. In case of an overflow, i.e. if LIM_MAX is reached, the counter restarts again at LIM_MIN.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If the lower limit value LIM_MIN is higher than the upper limit value LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO. In this case, the values for LIM_MIN and LIM_MAX are not forwarded to the DC541. The difference between LIM_MAX and LIM_MIN has to be at least twice the number (frequency) of counting pulses per DC541 cycle.

Example:

- Number of counting pulses (frequency) = 40 kHz = 40000 increments/s = 40 increments/ms
- Cycle time of the DC541 = 100 µs
- LIM_MIN = 0

-> Number of counting pulses per DC541 cycle: 40 increments/ms = 4 increments/100 µs
-> LIM_MAX > 8

Using input SET, the counter is set to the value CNT_SET. This value is kept as long as input SET = TRUE.

If the reference point approach is enabled at input EN_REF, the counter is set to the value of input CNT_SET when a rising edge occurs on channel C2 or C3.

Using input EN_TOUCH, a touch trigger measurement is enabled. This means: With the rising edge on channel C2 or C3, the counting value is stored and displayed at output CNT_TOUCH. The validity of CNT_TOUCH is indicated by output RDY_TOUCH. This functionality can be used to determine the counter value with regard to an external event. The results are increment accurate.

Only one function may be enabled at a time, either the reference point approach or the touch trigger measurement. If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, a corresponding error message is displayed at the outputs ERR/ERNO.

To initiate a new reference point approach or touch trigger measurement, a positive edge at the corresponding enabling input is necessary.

If the zero track of an incremental transmitter is wired to channel C2 or C3, no touch trigger measurement may be performed in the region of the reference cam!

The device DC541 must be configured as counting device (counter mode).

The block DC541_32BIT_CNT has an integrated visualization visuDC541_32BIT_CNT which can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description.

SLOT BYTE (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

CH BYTE (channel)

Input CH is used to select the input for the counter. Currently the only valid value is 0. The device occupies the inputs C0...C3. If an invalid value is entered at input CH or if the selected channel is not configured as a 32 bit counter (32BIT_CNT), the block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

EN_VISU BOOL (enable input in visualization)

If input EN_VISU = TRUE, it is also possible to control the block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_32BIT_CNT").

EN   BOOL   (enable)

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. The block is not processed if input EN = FALSE.

When the block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

SET   BOOL   (set)

If input SET = TRUE, the counter is set to the value of input CNT_SET and remains at this value as long as input SET = TRUE. Counting is enabled again with the occurrence of the falling edge at input SET.

EN_REF   BOOL   (enable reference)

A rising edge at input EN_REF enables the reference point approach. If EN_REF = TRUE, a rising edge at input C2 or C3 causes the actual value of the counter CNT_ACT to be set to the value of input CNT_SET.

The next measurement is again initiated by a rising edge at input EN_REF.

Only one function may be enabled at a time, either the reference point approach or the touch trigger measurement. If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, a corresponding error message is displayed at the outputs ERR/ERNO.

EN_TOUCH   BOOL   (enable touchtrigger)

A rising edge at input EN_TOUCH enables a touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at input C2 or C3 causes the block to store the actual counter value and to display this value at output CNT_TOUCH.

The next measurement is again initiated by a rising edge at input EN_TOUCH.

Only one function may be enabled at a time, either the reference point approach or the touch trigger measurement. If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, a corresponding error message is displayed at the outputs ERR/ERNO.

EN_UD   BOOL   (enable up/down)

Input EN_UD is used to select the counting mode of the 32 bit counter. The following applies:

EN_UD=FALSE: Connection of an incremental transmitter (track A/B offset by 90°)
EN_UD=TRUE: Up/down counter (C1: count up, C0: count down)

EN_LIM   BOOL   (enable limit)

Input EN_LIM is used to set the operating mode for the counter:

EN_LIM = FALSE Infinite counter (endless mode)
EN_LIM = TRUE   Limiting counter (limit mode)

Switching between the operating modes can also be done during running operation. Furthermore it is possible to change the upper limit value LIM_MAX and the lower limit value LIM_MIN during running operation. The upper limit value LIM_MAX always has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

In case of a rising edge at input EN_LIM (transition from endless mode to limit mode), the present counter value can change as follows depending on the values of the lower (LIM_MIN) and the higher limit value (LIM_MAX):

ACT_CNT < LIM_MIN	ACT_CNT := LIM_MIN
LIM_MIN <= ACT_CNT <= LIM_MAX	ACT_CNT := ACT_CNT (no change)
ACT_CNT > LIM_MAX	ACT_CNT := LIM_MIN

LIM_MAX DWORD (limit maximum)

Input LIM_MAX is used to set the upper limit value for the counter. The upper limit value LIM_MAX always has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

LIM_MIN DWORD (limit minimum)

Input LIM_MIN is used to set the lower limit value for the counter. The upper limit value LIM_MAX always has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

CNT_SET DWORD (count set)

Input CNT_SET is used to adjust the counter set value.

If input SET = TRUE, the counter is set to the value of input CNT_SET and remains at this value as long as input SET = TRUE. Counting is enabled again with the occurrence of the falling edge at input SET.

In case of a reference point approch (EN_REF = TRUE), the rising edge on channel C2 causes the actual value of the counter (ACT_CNT) to be adjusted to the set value CNT_SET.

A FALSE -> TRUE edge at input EN activates the status polling. If the value at input SLOT is not valid, processing is aborted and a corresponding error is displayed at output ERR/ERNO. The block outputs are updated as long as input EN = TRUE. The block processing has been completed successfully, if output DONE changes to TRUE. During the processing of a request, state changes at input EN are recognized but not evaluated.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages displayed at ERNO is explained in a separate table "Error messages of the block libraries".

RDY_REF BOOL (ready reference)

Output RDY_REF displays the ready message of the reference point approach.

RDY_TOUCH BOOL (ready touchtrigger)

Output RDY_TOUCH displays the ready message of the touch trigger measurement.

MAX_LIM BOOL (maximum limit)

In the infinite counter operating mode (endless mode), output MAX_LIM indicates whether the actual counter value is higher than the value set at input LIM_MAX.

In the limiting counter operating mode (limit mode), output MAX_LIM is FALSE.

MIN_LIM BOOL (minimum limit)

In the infinite counter operating mode (endless mode), output MIN_LIM indicates whether the actual counter value is lower than the value set at input LIM_MIN.

In the limiting counter operating mode (limit mode), output MIN_LIM is FALSE.

CNT_ACT DWORD (count actual)

Output CNT_ACT displays the actual counter value of the count-up counter.

CNT_TOUCH DWORD (count touchtrigger)

Output CNT_TOUCH displays the result of the touch trigger measurement, i.e. the actual counter value at the moment of the occurrence of the rising edge at input C2 after initiation of a touch trigger measurement (by a rising edge at input EN_TOUCH).

Function call in IL

```
CAL   DC541Cnt32Bit (
      SLOT    := DC541Cnt32Bit_SLOT,
      CH      := DC541Cnt32Bit_CH,
      EN_VISU := DC541Cnt32Bit_EN_VISU,
      EN      := DC541Cnt32Bit_EN,
      SET     := DC541Cnt32Bit_SET,
      EN_REF  := DC541Cnt32Bit_EN_REF,
      EN_TOUCH:= DC541Cnt32Bit_EN_TOUCH,
      EN_UD   := DC541Cnt32Bit_EN_UD,
      EN_LIM  := DC541Cnt32Bit_EN_LIM,
      LIM_MAX := DC541Cnt32Bit_LIM_MAX,
      LIM_MIN := DC541Cnt32Bit_LIM_MIN,
      CNT_SET := DC541Cnt32Bit_CNT_SET)


LD    DC541Cnt32Bit.DONE
ST    DC541Cnt32Bit_DONE

LD    DC541Cnt32Bit.ERR
ST    DC541Cnt32Bit_ERR

LD    DC541Cnt32Bit.ERNO
ST    DC541Cnt32Bit_ERNO

LD    DC541Cnt32Bit.RDY_REF
ST    DC541Cnt32Bit_RDY_REF

LD    DC541Cnt32Bit.RDY_TOUCH
ST    DC541Cnt32Bit_RDY_TOUCH

LD    DC541Cnt32Bit.MAX_LIM
ST    DC541Cnt32Bit_MAX_LIM

LD    DC541Cnt32Bit.MIN_LIM
ST    DC541Cnt32Bit_MIN_LIM
```

```
LD     DC541Cnt32Bit.CNT_ACT
ST     DC541Cnt32Bit_CNT_ACT

LD     DC541Cnt32Bit.CNT_TOUCH
ST     DC541Cnt32Bit_CNT_TOUCH
```

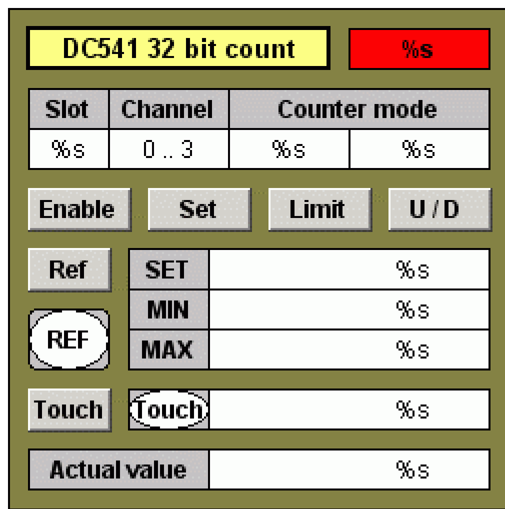
Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541Cnt32Bit (SLOT := DC541Cnt32Bit_SLOT,
      CH       := DC541Cnt32Bit_CH,
      EN_VISU  := DC541Cnt32Bit_EN_VISU,
      EN       := DC541Cnt32Bit_EN,
      SET      := DC541Cnt32Bit_SET,
      EN_REF   := DC541Cnt32Bit_EN_REF,
      EN_TOUCH:= DC541Cnt32Bit_EN_TOUCH,
      EN_UD    := DC541Cnt32Bit_EN_UD,
      EN_LIM   := DC541Cnt32Bit_EN_LIM,
      LIM_MAX  := DC541Cnt32Bit_LIM_MAX,
      LIM_MIN  := DC541Cnt32Bit_LIM_MIN,
      CNT_SET  := DC541Cnt32Bit_CNT_SET);


DC541Cnt32Bit_DONE      := DC541Cnt32Bit.DONE;
DC541Cnt32Bit_ERR       := DC541Cnt32Bit.ERR;
DC541Cnt32Bit_ERNO      := DC541Cnt32Bit.ERNO;
DC541Cnt32Bit_RDY_REF   := DC541Cnt32Bit.RDY_REF;
DC541Cnt32Bit_RDY_TOUCH := DC541Cnt32Bit.RDY_TOUCH;
DC541Cnt32Bit_MAX_LIM   := DC541Cnt32Bit.MAX_LIM;
DC541Cnt32Bit_MIN_LIM   := DC541Cnt32Bit.MIN_LIM;
DC541Cnt32Bit_CNT_ACT   := DC541Cnt32Bit.CNT_ACT;
DC541Cnt32Bit_CNT_TOUCH := DC541Cnt32Bit.CNT_TOUCH;
```

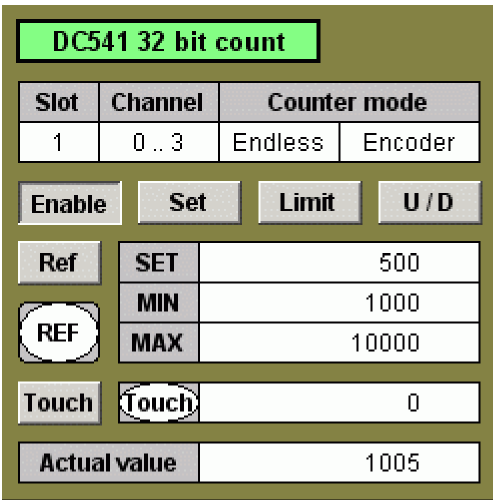
Integrated visualization of block DC541_32BIT_CNT

The block DC541_32BIT_CNT provides an integrated visualization visuDC541_32BIT_CNT which can be included and instantiated in other visualizations. In offline mode of the Control Builder, the visualization looks as follows:



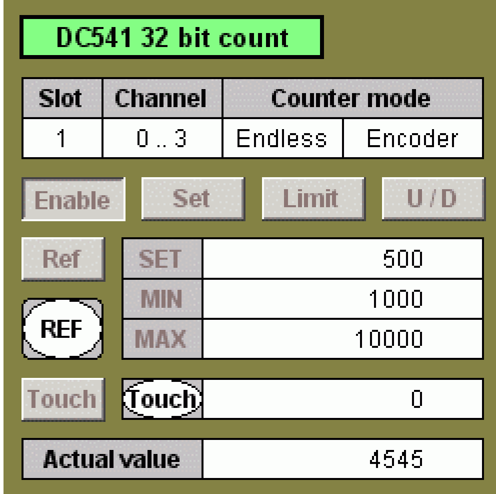
How to insert a visualization into a project and how to configure it is described at the beginning of the block library description (refer to Integrated visualization).

In online mode of the Control Builder, the visualization looks like in the following example, if EN_VISU is set to TRUE:

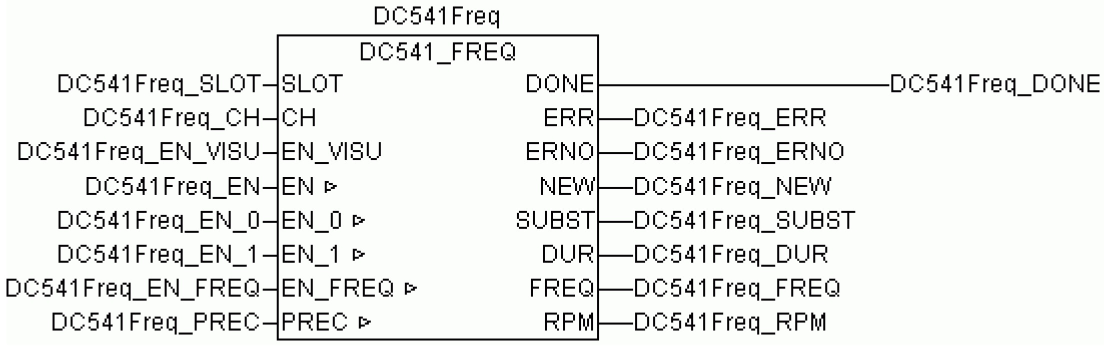


If an error occurred during block processing, the error number is displayed in the top right until EN becomes FALSE.

If input EN_VISU = FALSE, the inputs cannot be modified using the visualization. The corresponding control elements are then displayed in gray:



DC541_FREQ Time and frequency measurement



The block DC541_FREQ is used to measure times, frequencies and rotational speeds with a resolution of 100 µs.

Block data

Available as of PLC runtime system:	V1.1.3	Remark:
Available as of DC541 firmware:	V1.2	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_FREQ	Instance name
SLOT	Input	BYTE	Slot of the DC541
CH	Input	BYTE	Input selection C0...C7
EN_VISU	Input	BOOL	Enabling of control via the integrated visualization of block visuDC541_FREQ
EN	Input/Output	BOOL	Enabling of the block processing
EN_0	Input/Output	BOOL	Measurement: Negative edges
EN_1	Input/Output	BOOL	Measurement: Positive edges
EN_FREQ	Input/Output	BOOL	= TRUE: Enabling of frequency and rotational speed calculation
PREC	Input/Output	WORD	Measurement accuracy
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NEW	Output	BOOL	Indication that a new measurement is available
SUBST	Output	BOOL	Indication that calculation has been performed using substitute values
DUR	Output	DWORD	Measured time in [µs]
FREQ	Output	LREAL	Measured frequency in [Hz]
RPM	Output	LREAL	Measured speed of rotation in [rpm]

Description

The block DC541_FREQ is used to measure times, frequencies and rotational speeds with a resolution of 100 µs.

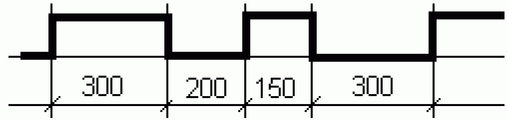
It is able to measure frequencies from 0 to 2000 Hz (2 kHz). In order to obtain a precise measurement of frequencies > 50 Hz, a correspondingly high accuracy setting has to be chosen. It is recommended to use an accuracy of PREC = 1000, i.e. 0.001.

This block has to be called cyclically, one time per second at least.

The inputs EN_0, EN_1 and EN_FREQ are used to determine the edges to be measured. If input EN_FREQ = TRUE, the frequency and the rotational speed are calculated in addition to the time measurement.

EN_0	EN_1	EN_FREQ	Edges measured	FREQ/RPM
FALSE	FALSE	TRUE	No measurement is performed.	yes
FALSE	TRUE	TRUE	Measurement of time between two rising edges	yes
TRUE	FALSE	TRUE	Measurement of time between two falling edges	yes
TRUE	TRUE	TRUE	Measurement of time between any two edges	yes
FALSE	FALSE	FALSE	No measurement is performed.	no
FALSE	TRUE	FALSE	Measurement of time between the falling edge and the subsequent rising edge	no
TRUE	FALSE	FALSE	Measurement of time between the rising edge and the subsequent falling edge	no
TRUE	TRUE	FALSE	Measurement of time between any two edges	no

The following example shows the different time measurement results depending on the values applied to the inputs EN_0, EN_1 and EN_FREQ.



EN_0	EN_1	EN_FREQ	Time measurement (duration DUR) in [µs]			
			1	2	3	4
FALSE	FALSE	TRUE	0	0	0	0
FALSE	TRUE	TRUE	-	500	-	450
TRUE	FALSE	TRUE	-	-	350	-
TRUE	TRUE	TRUE	300	200	150	300
FALSE	FALSE	FALSE	0	0	0	0
FALSE	TRUE	FALSE	300	-	150	-
TRUE	FALSE	FALSE	-	200	-	300
TRUE	TRUE	FALSE	300	200	150	300

The output NEW indicates that new measurement results are available.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency measurement.

The block DC541_FREQ has an integrated visualization visuDC541_FREQ which can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_FREQ").

SLOT BYTE (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

CH BYTE (channel)

Input CH is used to select the channel for time and frequency measurement. Valid values are 0...7 for the inputs C0...C7. If an invalid value is specified at input CH or if the selected channel is not configured as frequency measurement, the block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

EN_VISU BOOL (enable input in visualization)

If input EN_VISU = TRUE, it is also possible to control the block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_FREQ").

EN BOOL (enable)

In order to enable pulse counting for input CH, input EN has to be continuously provided with TRUE. If input EN = FALSE, the block is not processed and the pulses at input CH are lost.

When the block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

EN_0 BOOL (enable 0)

Input EN_0 is used to determine whether falling edges are considered for measurement:

EN_0 = TRUE: Measurement considers falling edges.
EN_0 = FALSE: Measurement does not consider falling edges.

EN_1 BOOL (enable 1)

Input EN_1 is used to determine whether rising edges are considered for measurement:

EN_1 = TRUE: Measurement considers rising edges.
EN_1 = FALSE: Measurement does not consider rising edges.

EN_FREQ BOOL (enable frequency)

EN_FREQ = FALSE: Measurement of time between positive or negative edges
EN_FREQ = TRUE: Measurement of time between edges and calculation of frequency and speed of rotation

PREC WORD (precision)

Input PREC is used to specify the demanded accuracy of the measurement.

PREC = 10 corresponds to an accuracy of 0.1 (one tenth)

PREC = 100 corresponds to an accuracy of 0.01 (one hundredth)

PREC = 1000 corresponds to an accuracy of 0.001 (one thousandth)

Depending on the cycle time and the demanded accuracy, the DC541 extends its measuring time and counts multiple pulses. This setting is only used in the frequency measurement operating mode (EN_FREQ = TRUE). Then, the displayed measurement values correspond to the measured average value. This reduces jitter and latency effects resulting in improved accuracy.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

NEW BOOL (new)

The output NEW indicates that new measurement results are available.

NEW = TRUE: New measurement results are available.
NEW = FALSE: No new measurement results are available.

SUBST BOOL (substitute)

Output SUBST indicates whether the output values are based on new measurement values or calculated using substitute values. Without the substitute value, the block would still display a short time and a high speed of rotation in case of a decrease of the frequency, since no new measurement is initiated without a new signal edge. If SUBST = TRUE, the extended distance between the edges is extrapolated. PREC > 0 is an assumption for the calculation using a substitute value.

SUBST=FALSE: New measurement values
SUBST=TRUE: Calculation using substitute values

DUR DWORD (duration)

Output DUR displays the measured time in [µs].

FREQ LREAL (frequency)

If input EN_FREQ = TRUE, output FREQ displays the frequency in [Hz] calculated from the measured time.

If input EN_FREQ is FALSE, output FREQ is 0.

RPM LREAL (revolutions per minute)

If input EN_FREQ = TRUE, output RPM displays the speed of rotation in [rpm] calculated from the measured time.

If input EN_FREQ is FALSE, output RPM is 0.

Function call in IL

```
CAL    DC541Freq(
       SLOT    := DC541Freq_SLOT,
       CH      := DC541Freq_CH,
       EN_VISU := DC541Freq_EN_VISU,
       EN      := DC541Freq_EN,
       EN_0    := DC541Freq_EN_0,
       EN_1    := DC541Freq_EN_1,
       EN_FREQ := DC541Freq_EN_FREQ,
       PREC    := DC541Freq_PREC)

LD     DC541Freq.DONE
ST     DC541Freq_DONE

LD     DC541Freq.ERR
ST     DC541Freq_ERR

LD     DC541Freq.ERNO
ST     DC541Freq_ERNO

LD     DC541Freq.NEW
ST     DC541Freq_NEW

LD     DC541Freq.SUBST
ST     DC541Freq_SUBST

LD     DC541Freq.DUR
ST     DC541Freq_DUR

LD     DC541Freq.FREQ
ST     DC541Freq_FREQ

LD     DC541Freq.RPM
ST     DC541Freq_RPM
```

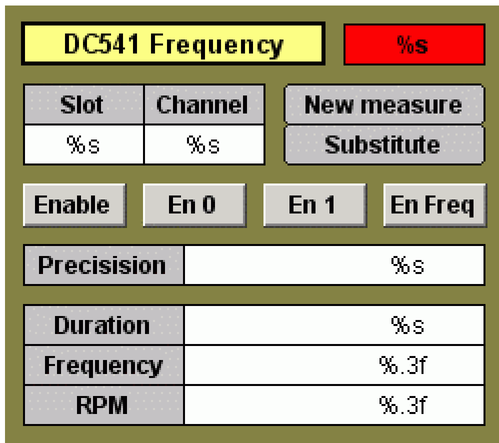
Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541Freq(SLOT       := DC541Freq_SLOT,
          CH         := DC541Freq_CH,
          EN_VISU    := DC541Freq_EN_VISU,
          EN         := DC541Freq_EN,
          EN_0       := DC541Freq_EN_0,
          EN_1       := DC541Freq_EN_1,
          EN_FREQ    := DC541Freq_EN_FREQ,
          PREC       := DC541Freq_PREC);

DC541Freq_DONE       := DC541Freq.DONE;
DC541Freq_ERR        := DC541Freq.ERR;
DC541Freq_ERNO       := DC541Freq.ERNO;
DC541Freq_NEW        := DC541Freq.NEW;
DC541Freq_SUBST      := DC541Freq.SUBST;
DC541Freq_DUR        := DC541Freq.DUR;
DC541Freq_FREQ       := DC541Freq.FREQ;
DC541Freq_RPM        := DC541Freq.RPM;
```

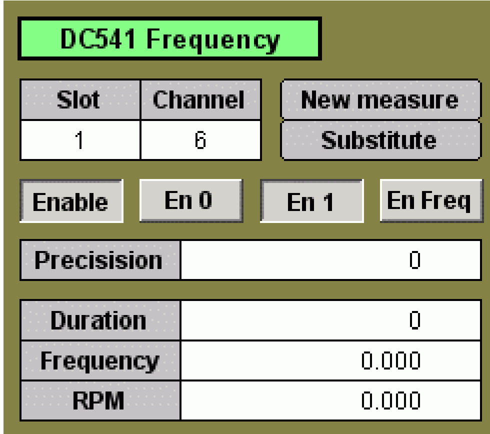
Integrated visualization of block DC541_FREQ

The block DC541_FREQ provides an integrated visualization visuDC541_FREQ which can be included and instantiated in other visualizations. In offline mode of the Control Builder, the visualization looks as follows:



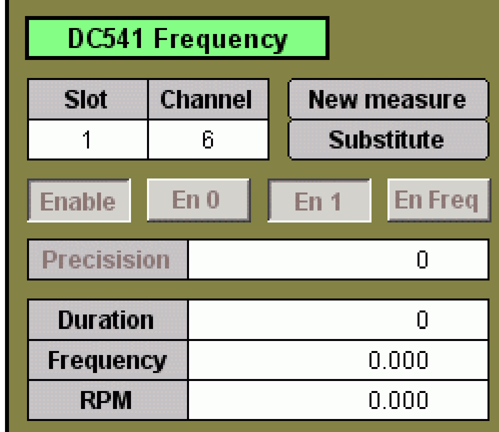
How to insert a visualization into a project and how to configure it is described at the beginning of the block library description (refer to Integrated visualization).

In online mode of the Control Builder, the visualization looks like in the following example, if EN_VISU is set to TRUE:

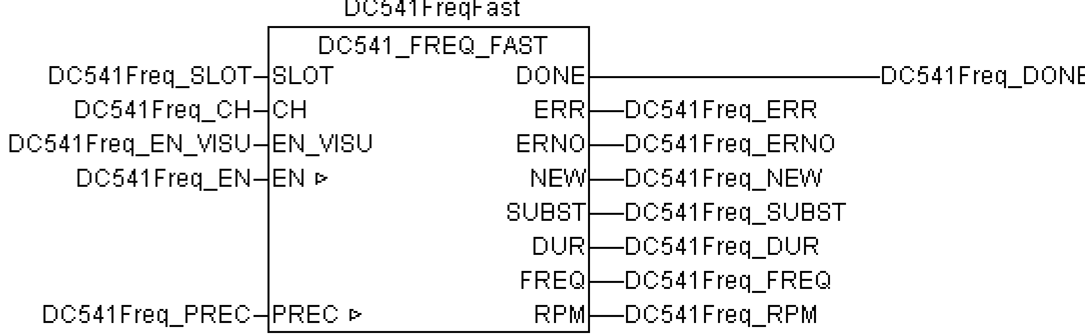


If an error occurred during block processing, the error number is displayed in the top right until EN becomes FALSE.

If input EN_VISU = FALSE, the inputs cannot be modified using the visualization. The corresponding control elements are then displayed in gray:



DC541_FREQ_FAST Fast time and frequency measurement



The block DC541_FREQ_FAST is used to measure times, frequencies and rotational speeds with a resolution of 1 µs.

Block data

Available as of PLC runtime system:	V1.2	Remark:
Available as of DC541 firmware:	V1.3	
Included in library:	DC541_AC500_V12.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_FREQ_FAST	Instance name
SLOT	Input	BYTE	Slot of the DC541
CH	Input	BYTE	Input selection C0...C7
EN_VISU	Input	BOOL	Enabling of control via the integrated visualization of block visuDC541_FREQ_FAST
EN	Input/Output	BOOL	Enabling of the block processing
PREC	Input/Output	WORD	Measurement accuracy
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
NEW	Output	BOOL	Indication that a new measurement is available
SUBST	Output	BOOL	Indication that calculation has been performed using substitute values
DUR	Output	DWORD	Measured time in [µs]
FREQ	Output	LREAL	Measured frequency in [Hz]
RPM	Output	LREAL	Measured speed of rotation in [rpm]

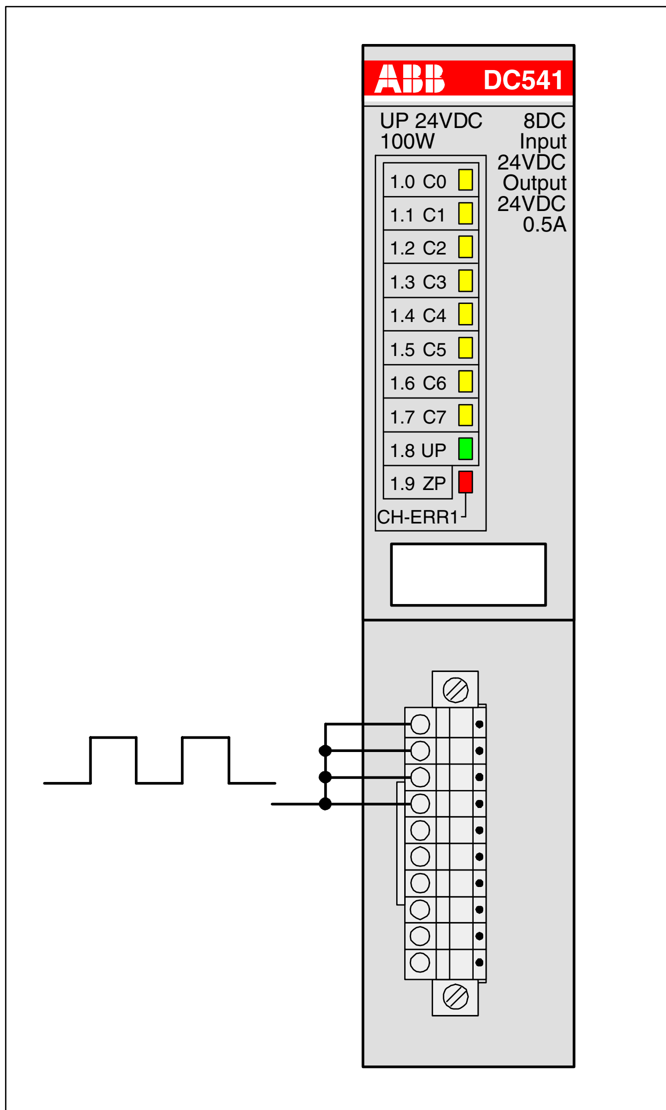
Description

The block DC541_FREQ is used to measure times, frequencies and rotational speeds with a resolution of 1 µs.

It is able to measure frequencies from 0 to 50000 Hz (50 kHz).

The function block works just like DC541_FREQ, but it reaches a resolution of 1us. The higher resolution, however, results in some important restrictions:

1. The function block is only allowed once per DC541 and only for channel 0.
2. It occupies the resources of the first 4 inputs, which cannot be used for other purposes.
3. The inputs C0..C3 must be connected in parallel.



This block has to be called cyclically, one time per second at least.

It always measures the time and frequency from one positive edge to the next positive edge.

The output NEW indicates that new measurement results are available.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency measurement.

The block DC541_FREQ_FAST has an integrated visualization visuDC541_FREQ_FAST which can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_FREQ_FAST").

SLOT BYTE (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

CH BYTE (channel)

Input CH is used to select the channel for time and frequency measurement. Valid values are 0...7 for the inputs C0...C7. If an invalid value is specified at input CH or if the selected channel is not configured as frequency measurement, the block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

EN_VISU BOOL (enable input in visualization)

If input EN_VISU = TRUE, it is also possible to control the block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_FREQ_FAST").

EN BOOL (enable)

In order to enable pulse counting for input CH, input EN has to be continuously provided with TRUE. If input EN = FALSE, the block is not processed and the pulses at input CH are lost.

When the block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

PREC WORD (precision)

Input PREC is used to specify the demanded accuracy of the measurement.

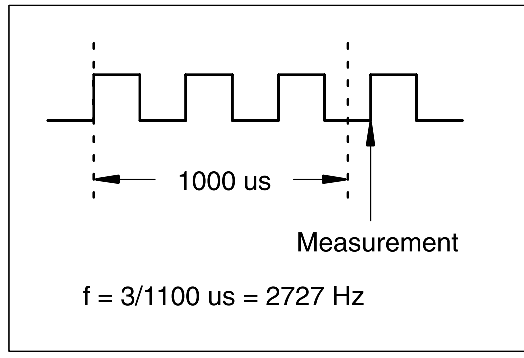
PREC = 10 corresponds to an accuracy of 0.1 (one tenth)

PREC = 100 corresponds to an accuracy of 0.01 (one hundredth)

PREC = 1000 corresponds to an accuracy of 0.001 (one thousandth)

Depending on the cycle time and the demanded accuracy, the DC541 extends its measuring time and counts multiple pulses. This is only used in operating mode "frequency measurement" (EN_FREQ = TRUE). Then, the displayed measurement values correspond to the measured average value. This reduces jitter and latency effects resulting in improved accuracy.

As the resolution for measuring is 1 µs, the precision corresponds directly to the measuring time in µs. For times < 100 µs, the device will just calculate the values as fast as possible, which will be the cycle time of the device, around 100 µs. For higher precision, the device will use the pulses per measuring time. It will measure the pulses and the time, with the time on a 1µs resolution.



DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

NEW BOOL (new)

The output NEW indicates that new measurement results are available.

NEW = TRUE: New measurement results are available.
NEW = FALSE: No new measurement results are available.

SUBST BOOL (substitute)

Output SUBST indicates whether the output values are based on new measurement values or calculated using substitute values. Without the substitute value, the block would still display a short time and a high speed of rotation in case of a decrease of the frequency, since no new measurement is initiated without a new signal edge. If SUBST = TRUE, the extended distance between the edges is extrapolated. PREC > 0 is an assumption for the calculation using a substitute value.

SUBST=FALSE: New measurement values
SUBST=TRUE: Calculation using substitute values

DUR DWORD (duration)

Output DUR displays the measured time in [µs].

FREQ LREAL (frequency)

If input EN_FREQ = TRUE, output FREQ displays the frequency in [Hz] calculated from the measured time.

If input EN_FREQ is FALSE, output FREQ is 0.

RPM LREAL (revolutions per minute)

If input EN_FREQ = TRUE, output RPM displays the speed of rotation in [rpm] calculated from the measured time.

If input EN_FREQ is FALSE, output RPM is 0.

Function call in IL

```
CAL    DC541FreqFast(
       SLOT    := DC541FreqFast_SLOT,
       CH      := DC541FreqFast_CH,
       EN_VISU := DC541FreqFast_EN_VISU,
       EN      := DC541FreqFast_EN,
       PREC    := DC541FreqFast_PREC)

LD     DC541FreqFast.DONE
ST     DC541FreqFast_DONE

LD     DC541FreqFast.ERR
ST     DC541FreqFast_ERR

LD     DC541FreqFast.ERNO
ST     DC541FreqFast_ERNO

LD     DC541FreqFast.NEW
ST     DC541FreqFast_NEW

LD     DC541FreqFast.SUBST
ST     DC541FreqFast_SUBST

LD     DC541FreqFast.DUR
ST     DC541FreqFast_DUR

LD     DC541FreqFast.FREQ
ST     DC541FreqFast_FREQ

LD     DC541FreqFast.RPM
ST     DC541FreqFast_RPM
```

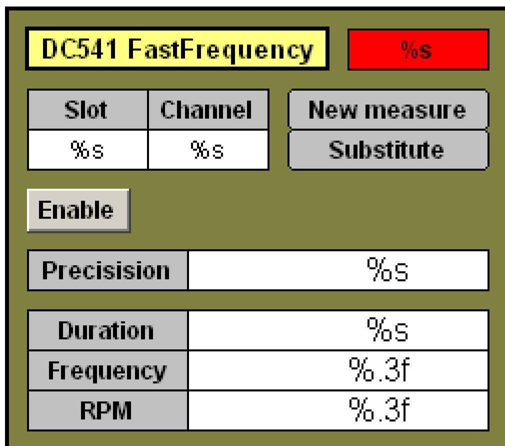
Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541FreqFast(SLOT      := DC541FreqFast_SLOT,
              CH        := DC541FreqFast_CH,
              EN_VISU   := DC541FreqFast_EN_VISU,
              EN        := DC541FreqFast_EN,
              EN_0      := DC541FreqFast_EN_0,
              EN_1      := DC541FreqFast_EN_1,
              EN_FREQ   := DC541FreqFast_EN_FREQ,
              PREC      := DC541FreqFast_PREC);

DC541FreqFast_DONE      := DC541FreqFast.DONE;
DC541FreqFast_ERR       := DC541FreqFast.ERR;
DC541FreqFast_ERNO      := DC541FreqFast.ERNO;
DC541FreqFast_NEW       := DC541FreqFast.NEW;
DC541FreqFast_SUBST     := DC541FreqFast.SUBST;
DC541FreqFast_DUR       := DC541FreqFast.DUR;
DC541FreqFast_FREQ      := DC541FreqFast.FREQ;
DC541FreqFast_RPM       := DC541FreqFast.RPM;
```

Integrated visualization of block DC541_FREQ_FAST

The block DC541_FREQ_FAST provides an integrated visualization visuDC541_FREQ_FAST which can be included and instantiated in other visualizations. In offline mode of the Control Builder, the visualization looks as follows:

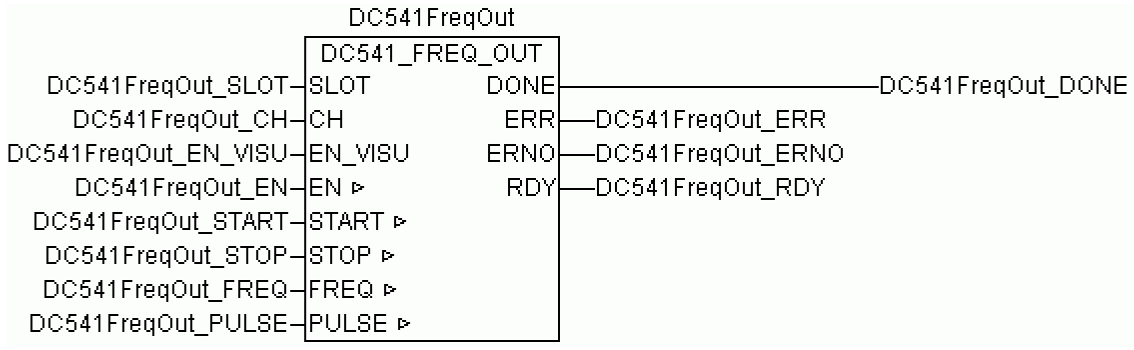


How to insert a visualization into a project and how to configure it is described at the beginning of the block library description (refer to Integrated visualization).

If an error occurred during block processing, the error number is displayed in the top right until EN becomes FALSE.

If input EN_VISU = FALSE, the inputs cannot be modified using the visualization. The corresponding control elements are then displayed in gray.

DC541_FREQ_OUT Frequency output



The block DC541_FREQ_OUT is used to output pulses of an adjustable frequency on one channel CH of the device DC541.

Block data

Available as of PLC runtime system:	V1.1.3	Remark:
Available as of DC541 firmware:	V1.2	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_FREQ_OUT	Instance name
SLOT	Input	BYTE	Slot of the DC541
CH	Input	BYTE	Output selection C0...C7
EN_VISU	Input	BOOL	Enabling of control via the integrated visualization of block visuDC541_FREQ_OUT
EN	Input/Output	BOOL	Enabling of the block processing
START	Input/Output	BOOL	Start of frequency output
STOP	Input/Output	BOOL	Termination of frequency output
FREQ	Input/Output	LREAL	Set value presetting for frequency [Hz]
PULSE	Input/Output	DWORD	Set value presetting for pulses: =0 - infinite >0 - number of pulses
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
RDY	Output	BOOL	Number of pulses, output if PULSE > 0

Description

The block DC541_FREQ_OUT is used to output pulses with a fixed frequency on one channel of the device DC541. It is able to output pulses with a frequency between 0.2 and 2.5 kHz. The pulse jitter depends on the cycle time of the DC541. The pulse length is always a multiple of the cycle time of the DC541.

In case of a presetting of PULSE = 0, the output of pulses is infinite. The pulse output is started with a positive edge at input START. The output is aborted if START = FALSE. A positive edge at input STOP interrupts the pulse output. The output is continued if STOP = FALSE.

If input PULSE > 0, the block outputs the number of pulses specified at input PULSE with the frequency specified at input FREQ on the channel specified at input CH. After the block has output the number of pulses specified at PULSE, the output RDY becomes TRUE.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency output.

The block DC541_FREQ_OUT has an integrated visualization visuDC541_FREQ_OUT which can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_FREQ_OUT").

SLOT BYTE (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

CH BYTE (channel)

Input CH is used to select the channel for the frequency output. Valid values are 0...7 for the outputs C0...C7. If an invalid value is specified at input CH or if the selected channel is not configured as frequency output, the block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

EN_VISU BOOL (enable input in visualization)

If input EN_VISU = TRUE, it is also possible to control the block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_FREQ_OUT").

EN BOOL (enable)

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. If input EN = FALSE, the block is not processed and the pulses at input CH are lost.

When the block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

START BOOL (start)

A positive edge at input START starts the frequency output. The frequency output is aborted if START = FALSE.

STOP BOOL (stop)

A positive edge at input STOP interrupts the frequency output. If a given number of pulses has to be output, the counter continues counting if STOP = FALSE. The STOP is not synchronized with the frequency.

FREQ LREAL (frequency)

Input FREQ is used to preset the frequency to be output in [Hz]. Valid values are 0.2 Hz to 2500.0 Hz (2.5 kHz).

PULSE DWORD (pulse)

Input PULSE is used to determine whether frequency output shall be performed endless or only for the specified number of pulses. The following applies:

PULSE=0: Endless output
PULSE>0: Output of the specified number of pulses

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

RDY BOOL (ready)

If PULSE is > 0, output RDY becomes TRUE after the specified number of pulses has been output.

Function call in IL

```
CAL   DC541FreqOut(
      SLOT    := DC541FreqOut_SLOT,
      CH      := DC541FreqOut_CH,
      EN_VISU := DC541FreqOut_EN_VISU,
      EN      := DC541FreqOut_EN,
      START   := DC541FreqOut_START,
      STOP    := DC541FreqOut_STOP,
      FREQ    := DC541FreqOut_FREQ,
      PULSE   := DC541FreqOut_PULSE)

LD    DC541FreqOut.DONE
ST    DC541FreqOut_DONE

LD    DC541FreqOut.ERR
ST    DC541FreqOut_ERR

LD    DC541FreqOut.ERNO
ST    DC541FreqOut_ERNO

LD    DC541FreqOut.RDY
ST    DC541FreqOut_RDY
```

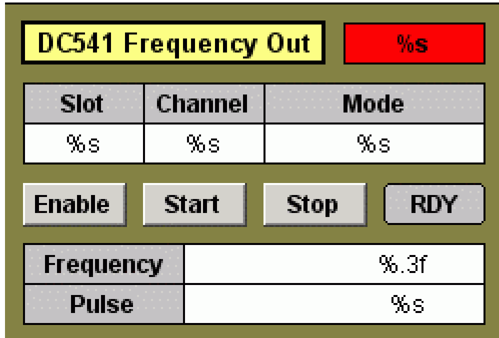
Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541FreqOut(SLOT       := DC541FreqOut_SLOT,
             CH         := DC541FreqOut_CH,
             EN_VISU    := DC541FreqOut_EN_VISU,
             EN         := DC541FreqOut_EN,
             START      := DC541FreqOut_START,
             STOP       := DC541FreqOut_STOP,
             FREQ       := DC541FreqOut_FREQ,
             PULSE      := DC541FreqOut_PULSE);

DC541FreqOut_DONE       := DC541FreqOut.DONE;
DC541FreqOut_ERR        := DC541FreqOut.ERR;
DC541FreqOut_ERNO       := DC541FreqOut.ERNO;
DC541FreqOut_RDY        := DC541FreqOut.RDY;
```

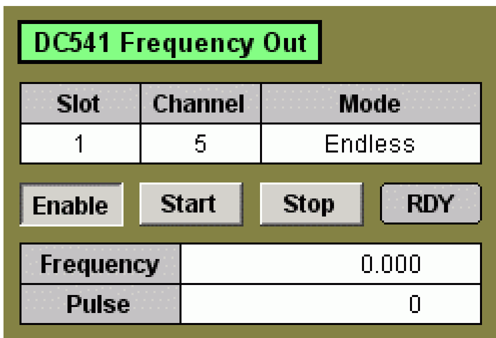
Integrated visualization of block DC541_FREQ_OUT

The block DC541_FREQ_OUT provides an integrated visualization visuDC541_FREQ_OUT which can be included and instantiated in other visualizations. In offline mode of the Control Builder, the visualization looks as follows:



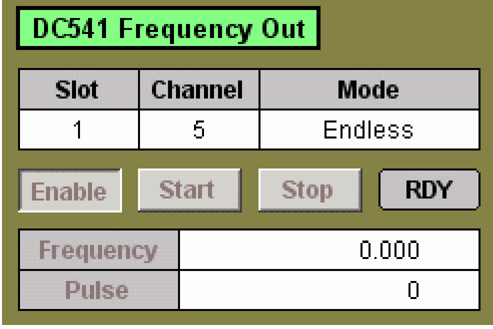
How to insert a visualization into a project and how to configure it is described at the beginning of the block library description (refer to Integrated visualization).

In online mode of the Control Builder, the visualization looks like in the following example, if EN_VISU is set to TRUE:

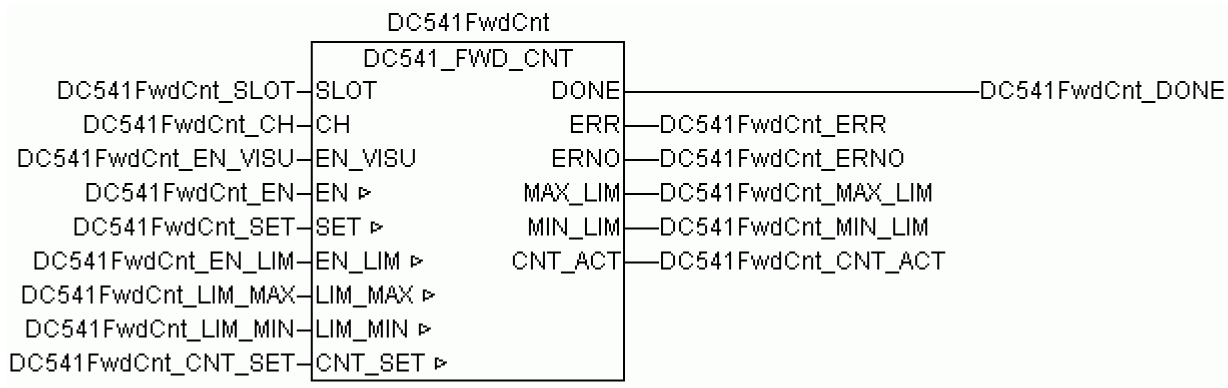


If an error occurred during block processing, the error number is displayed in the top right until EN becomes FALSE.

If input EN_VISU = FALSE, the inputs cannot be modified using the visualization. The corresponding control elements are then displayed in gray:



DC541_FWD_CNT 32 bit count-up counter



This block DC541_FWD_CNT is a 32 bit count-up counter.

Block data

Available as of PLC runtime system:	V1.1.3	Remark:
Available as of DC541 firmware:	V1.1	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_FWD_CNT	Instance name
SLOT	Input	BYTE	Slot of the DC541
CH	Input	BYTE	=0, channel number, currently channel C0 only
EN_VISU	Input	BOOL	Enabling of control via the integrated visualization of block visuDC541_FWD_CNT
EN	Input/Output	BOOL	Enabling of the block processing
SET	Input/Output	BOOL	Set input
EN_LIM	Input/Output	BOOL	Counting mode selection: Infinite counter or limiting counter
LIM_MAX	Input/Output	DWORD	Counter end value if EN_LIM = TRUE
LIM_MIN	Input/Output	DWORD	Counter start value if EN_LIM = TRUE
CNT_SET	Input/Output	DWORD	Counter set value
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
MAX_LIM	Output	BOOL	=TRUE: LIM_MAX reached (infinite counter)
MIN_LIM	Output	BOOL	=TRUE: LIM_MIN reached (infinite counter)
CNT_ACT	Output	DWORD	Actual counter value

Description

The block DC541_FWD_CNT provides a 32 bit count-up counter which is able to count a maximum frequency of 50 kHz at the inputs C0 and C1 or 5 kHz at the inputs C2...C7. In the DC541, the counter is implemented as a 16 bit counter. The actual counter value ACT_CNT is bulit inside the block by adding the counter differences that occur within the individual cycles. In order not to loose any counting pulses, the block has to be called cyclically with at least the following interval:

- Channel 0...1: 50 kHz max. -> 32767 / 50 = 655 ms
- Channel 2...7:   5 kHz max. -> 32767 / 5 = 6550 ms

Using the counter e.g. in a 100 ms task will prevent any loss of counting pulses.

The counter can be used in two operating modes:

- Infinite counter (endless mode)
- Limiting counter (limit mode)

The operating mode is selected at input EN_LIM.

If EN_LIM = FALSE, the counter operates as an infinite counter (endless mode). An overflow occurs corresponding to the 32 bit value at 16#FFFFFFFF = 4 294 967 295. In this mode, any exceeding of the limit value LIM_MAX or falling below the limit value LIM_MIN is displayed at the outputs MAX_LIM or MIN_LIM.

If EN_LIM = TRUE (limit mode), the counting range is between the limit values LIM_MIN and LIM_MAX. In case of an overflow, i.e. if LIM_MAX is reached, the counter restarts again at LIM_MIN.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If the lower limit value LIM_MIN is higher than the upper limit value LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

The device DC541 must be configured as counting device (counter mode).

The block DC541_FWD_CNT has an integrated visualization visuDC541_FWD_CNT which can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_FWD_CNT").

SLOT   BYTE   (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

CH   BYTE   (channel)

Input CH is used to select the input for the counter. Valid values are 0..7 for the channels C0...C7. If an invalid value is entered at input CH or if the selected channel is not configured as 32 bit count-up counter (FWD_CNT), the block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

EN_VISU   BOOL   (enable input in visualization)

If input EN_VISU = TRUE, it is also possible to control the block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_FWD_CNT").

EN BOOL (enable)

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. The block is not processed if input EN = FALSE.

When the block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

SET BOOL (set)

If input SET = TRUE, the counter is set to the value of input CNT_SET and remains at this value as long as input SET = TRUE. Counting is enabled again with the occurrence of the falling edge at input SET.

EN_LIM BOOL (enable limit)

Input EN_LIM is used to set the operating mode for the counter:

EN_LIM = FALSE Infinite counter (endless mode)
EN_LIM = TRUE Limiting counter (limit mode)

Switching between the operating modes can also be done during running operation. Furthermore it is possible to change the upper limit value LIM_MAX and the lower limit value LIM_MIN during running operation. The upper limit value LIM_MAX always has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

In case of a rising edge at input EN_LIM (transition from endless mode to limit mode), the actual counter value can change as follows depending on the values of the lower (LIM_MIN) and the higher limit value (LIM_MAX):

ACT_CNT < LIM_MIN ACT_CNT := LIM_MIN
The counter value is smaller than LIM_MIN and thus set to the value of LIM_MIN.

LIM_MIN <= ACT_CNT <= LIM_MAX ACT_CNT := ACT_CNT (no change)
The counter value is within the limit values LIM_MIN and LIM_MAX and not changed.

ACT_CNT > LIM_MAX + DIFF ACT_CNT := LIM_MIN (DIFF = change within the current cycle)
The counter value is higher than LIM_MAX for several cycles and thus set to the value of LIM_MIN.

ACT_CNT > LIM_MAX ACT_CNT := CNT_ACT - (LIM_MAX - LIM_MIN + 1)
The counter value has exceeded LIM_MAX during the current cycle. The value change of the counter during the last cylce is considered when setting the counter.

Prior to the comparison of the new counter value with LIM_MIN and LIM_MAX and, if necessary, the subsequent setting of the value ACT_CNT, the new counter value is always calculated first from the counter value during the last cycle + the counter value change during the current cycle (DIFF).

LIM_MAX DWORD (limit maximum)

Input LIM_MAX is used to set the upper limit value for the counter. The upper limit value LIM_MAX always has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

LIM_MIN DWORD (limit minimum)

Input LIM_MIN is used to set the lower limit value for the counter. The upper limit value LIM_MAX always has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

CNT_SET DWORD (count set)

Input CNT_SET is used to adjust the counter set value.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

MAX_LIM BOOL (maximum limit)

In the infinite counter operating mode (endless mode), output MAX_LIM indicates whether the actual counter value is higher than the value set at input LIM_MAX.

In the limiting counter operating mode (limit mode), output MAX_LIM is FALSE.

MIN_LIM BOOL (minimum limit)

In the infinite counter operating mode (endless mode), output MIN_LIM indicates whether the actual counter value is lower than the value set at input LIM_MIN.

In the limiting counter operating mode (limit mode), output MIN_LIM is FALSE.

CNT_ACT DWORD (count actual)

Output CNT_ACT displays the actual counter value of the count-up counter.

Function call in IL

```
CAL   DC541FwdCnt(
      SLOT    := DC541FwdCnt_SLOT,
      CH      := DC541FwdCnt_CH,
      EN_VISU := DC541FwdCnt_EN_VISU,
      EN      := DC541FwdCnt_EN,
      SET     := DC541FwdCnt_SET,
      EN_LIM  := DC541FwdCnt_EN_LIM,
      LIM_MAX := DC541FwdCnt_LIM_MAX,
      LIM_MIN := DC541FwdCnt_LIM_MIN,
      CNT_SET := DC541FwdCnt_CNT_SET)

LD    DC541FwdCnt.DONE
ST    DC541FwdCnt_DONE

LD    DC541FwdCnt.ERR
ST    DC541FwdCnt_ERR

LD    DC541FwdCnt.ERNO
ST    DC541FwdCnt_ERNO
```

```
LD      DC541FwdCnt.MAX_LIM
ST      DC541FwdCnt_MAX_LIM

LD      DC541FwdCnt.MIN_LIM
ST      DC541FwdCnt_MIN_LIM

LD      DC541FwdCnt.CNT_ACT
ST      DC541FwdCnt_CNT_ACT
```

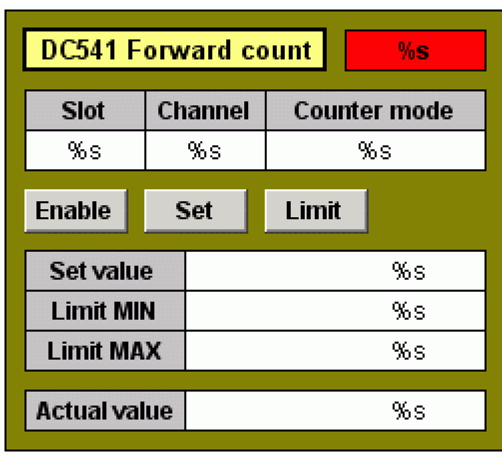
Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541FwdCnt(SLOT         := DC541FwdCnt_SLOT,
            CH           := DC541FwdCnt_CH,
            EN_VISU      := DC541FwdCnt_EN_VISU,
            EN           := DC541FwdCnt_EN,
            SET          := DC541FwdCnt_SET,
            EN_LIM       := DC541FwdCnt_EN_LIM,
            LIM_MAX      := DC541FwdCnt_LIM_MAX,
            LIM_MIN      := DC541FwdCnt_LIM_MIN,
            CNT_SET      := DC541FwdCnt_CNT_SET);

DC541FwdCnt_DONE         := DC541FwdCnt.DONE;
DC541FwdCnt_ERR          := DC541FwdCnt.ERR;
DC541FwdCnt_ERNO         := DC541FwdCnt.ERNO;
DC541FwdCnt_RDY_REF      := DC541FwdCnt.RDY_REF;
DC541FwdCnt_RDY_TOUCH    := DC541FwdCnt.RDY_TOUCH;
DC541FwdCnt_MAX_LIM      := DC541FwdCnt.MAX_LIM;
DC541FwdCnt_MIN_LIM      := DC541FwdCnt.MIN_LIM;
DC541FwdCnt_CNT_ACT      := DC541FwdCnt.CNT_ACT;
```

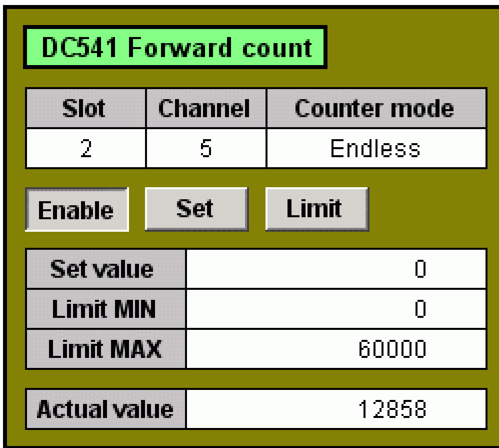
Integrated visualization of block DC541_FWD_CNT

The block DC541_FWD_CNT provides an integrated visualization visuDC541_FWD_CNT which can be included and instantiated in other visualizations. In offline mode of the Control Builder, the visualization looks as follows:



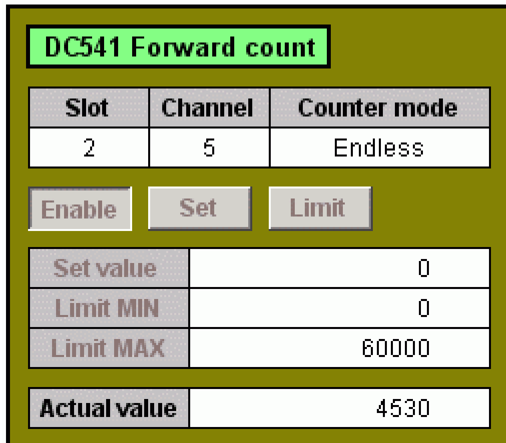
How to insert a visualization into a project and how to configure it is described at the beginning of the block library description (refer to Integrated visualization).

In online mode of the Control Builder, the visualization looks like in the following example, if EN_VISU is set to TRUE:

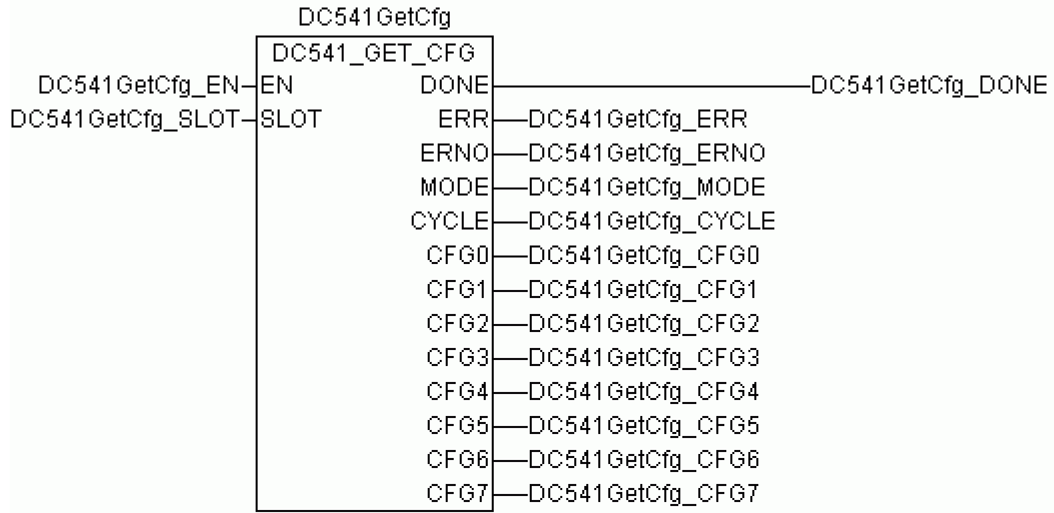


If an error occurred during block processing, the error number is displayed in the top right until EN becomes FALSE.

If input EN_VISU = FALSE, the inputs cannot be modified using the visualization. The corresponding control elements are then displayed in gray:



DC541_GET_CFG Output of DC541 configuration details



The block DC541_GET_CFG can be used to poll the configuration of the DC541 and its channels.

Block data

Available as of PLC runtime system:	V1.1.3	Remark:
Available as of DC541 firmware:	V1.1	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_GET_CFG	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot of the DC541
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
MODE	Output	BYTE	Display of the device's operating mode
CYCLE	Output	WORD	Display of the device's cycle time in [μs]
CFG0...CFG7	Output	BYTE	Display of the channel configuration for C0...C7

Description

The block DC541_GET_CFG is used to poll the actual configuration of the device DC541. The information is displayed as long as input EN is TRUE. The displayed information includes the operating mode and the cycle time of the device in [µs] as well as the configuration of the channels C0...C7.

The configuration for the device DC541 is done in the control system configuration of the Control Builder and described in the system technoloy documentation of the device DC541 (see also System technology DC541 / Device configuration for DC541)

EN   BOOL   (enable)

A FALSE -> TRUE edge at input EN activates the status polling. If the value at input SLOT is not valid, processing is aborted and a corresponding error is displayed at output ERR/ERNO. The block outputs are updated as long as input EN = TRUE. The block processing has been completed successfully, if output DONE changes to TRUE. During the processing of a request, state changes at input EN are recognized but not evaluated.

SLOT   BYTE   (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

DONE   BOOL   (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR   BOOL   (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO   WORD   (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

MODE BYTE   (mode)

Output MODE displays the actual operating mode of the device.
The following applies:

DC541_MODE_IO	= 16#11 = 17	- IO device (IO mode)
DC541_MODE_COUNT	= 16#22 = 34	- Counting device (counter mode)

CYCLE WORD (cycle time)

Output CYCLE displays the cycle time of the device in [µs]. The cycle time is set during the device configuration and can have the following values depending on the channel configuration:

IO device		50 µs
Counting device	1-2 functions	50 µs
	3-4 functions	100 µs
	5-8 functions	200 µs

"Functions" are:

- PWM	Pulse-width modulator
- FREQ	Time and frequency measurement
- FREQ_OUT	Frequency output
- 32BIT_CNT	32 bit encoder
- FWD_CNT	32 bit count-up counter
- LIMIT	Limit value monitoring for the 32 bit counter

CFG0...CFG7 BYTE (config C0...C7)

The outputs CFG0...CFG7 display the current channel configuration of the channels C0...C7. The values have the following meaning:

CFGx	Channel	Function
MODE = DC541_MODE_IO = 16#11 = 17		
0	0...7	Normal input
1	0...7	Normal output
255	0...7	Interrupt input
MODE = DC541_MODE_COUNT = 16#22 = 34		
0	0...7	Normal input
1	0...7	Normal output
2	0...7	Pulse-width modulator (PWM)
3	0...1	50 kHz count-up counter (FWD_CNT)
3	2...7	5 kHz count-up counter (FWD_CNT)
4	0...7	Time and frequency measurement (FREQ)
5	4...7	Limit values for 32 bit counter (LIMIT)
6	0	Bidirectional 32 bit counter (uses channels 0...3). The channels CFG1...CFG3 can be set as desired.
7	0...7	Frequency output

Function call in IL

```
CAL    DC541GetCfg (
       EN       := DC541GetCfg_EN,
       SLOT     := DC541GetCfg_SLOT)

LD     DC541GetCfg.DONE
ST     DC541GetCfg_DONE

LD     DC541GetCfg.ERR
ST     DC541GetCfg_ERR

LD     DC541GetCfg.ERNO
ST     DC541GetCfg_ERNO
```

```
LD    DC541GetCfg.MODE
ST    DC541GetCfg_MODE

LD    DC541GetCfg.CYCLE
ST    DC541GetCfg_CYCLE

LD    DC541GetCfg.CFG0
ST    DC541GetCfg_CFG0

LD    DC541GetCfg.CFG1
ST    DC541GetCfg_CFG1

LD    DC541GetCfg.CFG2
ST    DC541GetCfg_CFG2

LD    DC541GetCfg.CFG3
ST    DC541GetCfg_CFG3

LD    DC541GetCfg.CFG4
ST    DC541GetCfg_CFG4

LD    DC541GetCfg.CFG5
ST    DC541GetCfg_CFG5

LD    DC541GetCfg.CFG6
ST    DC541GetCfg_CFG6

LD    DC541GetCfg.CFG7
ST    DC541GetCfg_CFG7
```

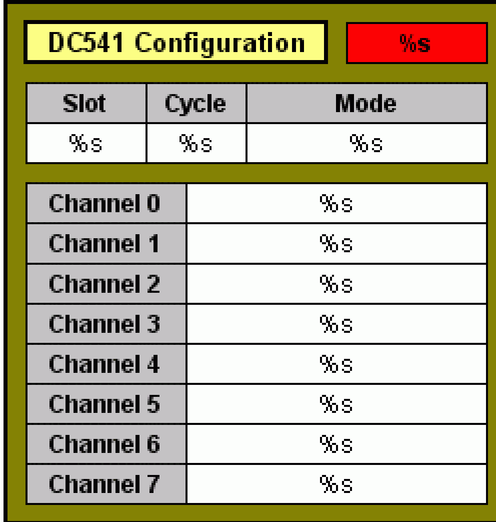
Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541GetCfg (EN   := DC541GetCfg_EN,
             SLOT := DC541GetCfg_SLOT);

DC541GetCfg_DONE  := DC541GetCfg.DONE;
DC541GetCfg_ERR   := DC541GetCfg.ERR;
DC541GetCfg_ERNO  := DC541GetCfg.ERNO;
DC541GetCfg_MODE  := DC541GetCfg.DIAG;
DC541GetCfg_CYCLE := DC541GetCfg.CYCLE;
DC541GetCfg_CFG0  := DC541GetCfg.CFG0;
DC541GetCfg_CFG1  := DC541GetCfg.CFG1;
DC541GetCfg_CFG2  := DC541GetCfg.CFG2;
DC541GetCfg_CFG3  := DC541GetCfg.CFG3;
DC541GetCfg_CFG4  := DC541GetCfg.CFG4;
DC541GetCfg_CFG5  := DC541GetCfg.CFG5;
DC541GetCfg_CFG6  := DC541GetCfg.CFG6;
DC541GetCfg_CFG7  := DC541GetCfg.CFG7;
```

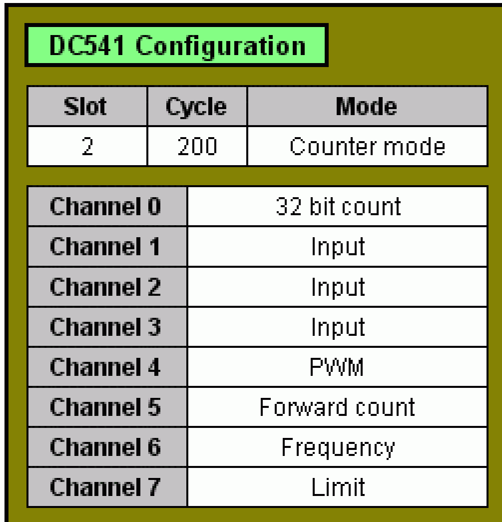
Integrated visualization of block DC541_GET_CFG

The block DC541_GET_CFG provides an integrated visualization visuDC541_GET_CFG which can be included and instantiated in other visualizations. In offline mode of the Control Builder, the visualization looks as follows:



How to insert a visualization into a project and how to configure it is described at the beginning of the block library description (refer to Integrated visualization).

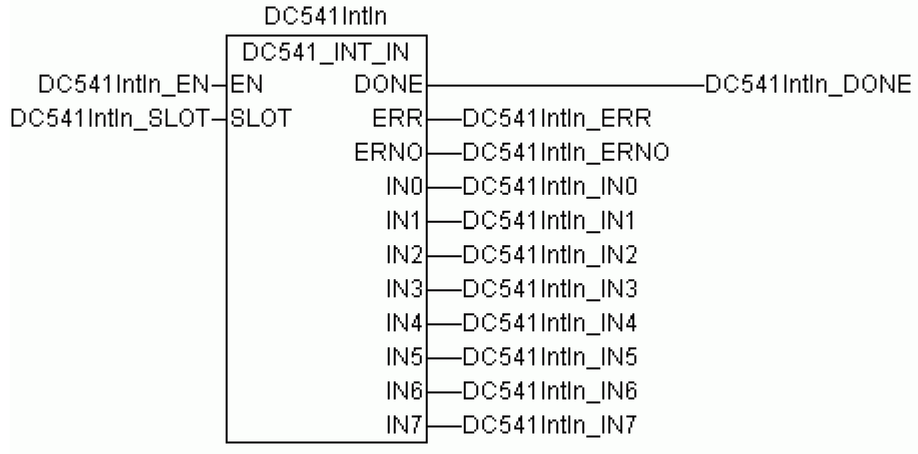
In online mode of the Control Builder, the visualization looks like in the following example:



The operating mode of the DC541 (Mode) and its channel configuration of the channels C0...C7 are displayed as plain text.

If an error occurred during block processing, the error number is displayed in the top right until EN becomes FALSE.

DC541_INT_IN Display the interrupt initiating source



The block DC541_INT_IN displays by which input (C0...C7) the interrupt was triggered.

Block data

Available as of PLC runtime system:	V1.1.2	Remark:
Available as of DC541 firmware:	V1.0	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_INT_IN	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot of the DC541
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
IN0...IN7	Output	BOOL	=TRUE: The corresponding input has triggered the interrupt.

Description

Using the block DC541_INT_IN, the interrupt program can be polled which inputs (C0...C7) triggered interrupts since the last block calling. The corresponding outputs IN0...IN1 are set to TRUE.

The block DC541_INT_IN can only be used if the device is configured as IO device.

EN BOOL (enable)

Input EN has to be continuously set to TRUE.

When the block is called for the first time, input SLOT is checked for validity and the corresponding device is checked for correct configuration in the operating mode "IO mode". If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO. The block outputs are updated as long as input EN = TRUE.

SLOT BYTE (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

IN0...IN7 BOOL (input C0...C7)

The outputs IN0...IN7 display which inputs triggered an interrupt since the last block calling.

Function call in IL

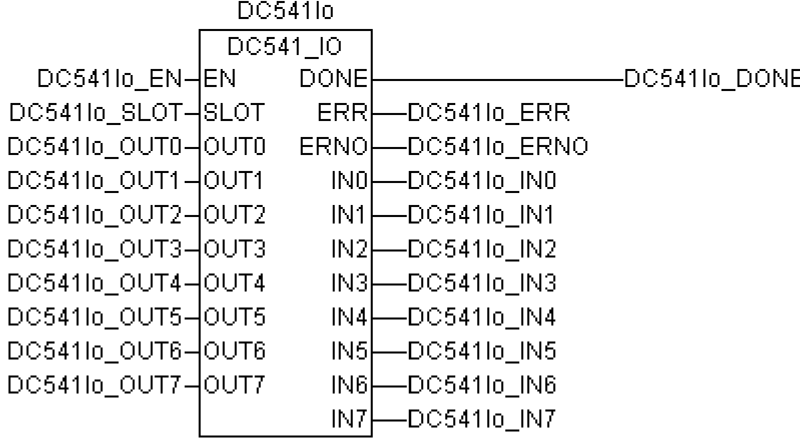
```
CAL    DC541IntIn (
       EN       := DC541IntIn_EN,
       SLOT     := DC541IntIn_SLOT)

LD     DC541IntIn.DONE
ST     DC541IntIn_DONE

LD     DC541IntIn.ERR
ST     DC541IntIn_ERR

LD     DC541IntIn.ERNO
ST     DC541IntIn_ERNO

LD     DC541IntIn.IN0
ST     DC541IntIn_IN0

LD     DC541IntIn.IN1
ST     DC541IntIn_IN1

LD     DC541IntIn.IN2
ST     DC541IntIn_IN2

LD     DC541IntIn.IN3
ST     DC541IntIn_IN3

LD     DC541IntIn.IN4
ST     DC541IntIn_IN4

LD     DC541IntIn.IN5
ST     DC541IntIn_IN5

LD     DC541IntIn.IN6
ST     DC541IntIn_IN6

LD     DC541IntIn.IN7
ST     DC541IntIn_IN7
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541IntIn(EN   := DC541IntIn_EN,
           SLOT := DC541IntIn_SLOT);

DC541IntIn_DONE := DC541IntIn.DONE;
DC541IntIn_ERR  := DC541IntIn.ERR;
DC541IntIn_ERNO := DC541IntIn.ERNO;
DC541IntIn_IN0  := DC541IntIn.IN0;
DC541IntIn_IN1  := DC541IntIn.IN1;
DC541IntIn_IN2  := DC541IntIn.IN2;
DC541IntIn_IN3  := DC541IntIn.IN3;
DC541IntIn_IN4  := DC541IntIn.IN4;
DC541IntIn_IN5  := DC541IntIn.IN5;
DC541IntIn_IN6  := DC541IntIn.IN6;
DC541IntIn_IN7  := DC541IntIn.IN7;
```

DC541_IO Reading/writing the inputs and outputs of the DC541



The block DC541_IO is used to read and write the inputs and outputs of the DC541.

Block data

Available as of PLC runtime system:	V1.1.2	Remark:
Available as of DC541 firmware:	V1.0	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_IO	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot of the DC541
OUT0...OUT7	Input	BOOL	Set value for the outputs of the channels C0...C7
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
IN0...IN7	Output	BOOL	Actual value of the inputs of the channels C0...C7

Description

The block DC541_IO is used to read the "normal" inputs and to write the outputs of the DC541. Reading and writing the inputs and outputs is performed as long as input EN is TRUE. This block can be used for both operating modes, i.e. when the device is configured as IO device and when it is configured as counting device (counter mode).

EN BOOL (enable)

A FALSE -> TRUE edge at input EN activates the status polling. If the value at input SLOT is not valid, processing is aborted and a corresponding error is displayed at output ERR/ERNO. The inputs and outputs of the DC541 are processed as long as input EN = TRUE. The block processing has been completed successfully, if output DONE changes to TRUE. During the processing of a request, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

OUT0...OUT7 BOOL (output C0...C7)

The inputs OUT0...OUT7 are used to specify the set status of the outputs C0...C7. Only channels that are configured as outputs are actually written.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

IN0...IN7 BOOL (input C0...C7)

The outputs IN0...IN7 display the actual status of the channels C0...C7.

Function call in IL

```
CAL   DC541Io (
      EN      := DC541Io_EN,
      SLOT    := DC541Io_SLOT,
      OUT0    := DC541Io_OUT0,
      OUT1    := DC541Io_OUT1,
      OUT2    := DC541Io_OUT2,
      OUT3    := DC541Io_OUT3,
      OUT4    := DC541Io_OUT4,
      OUT5    := DC541Io_OUT5,
      OUT6    := DC541Io_OUT6,
      OUT7    := DC541Io_OUT7)

LD    DC541Io.DONE
ST    DC541Io_DONE

LD    DC541Io.ERR
ST    DC541Io_ERR
```

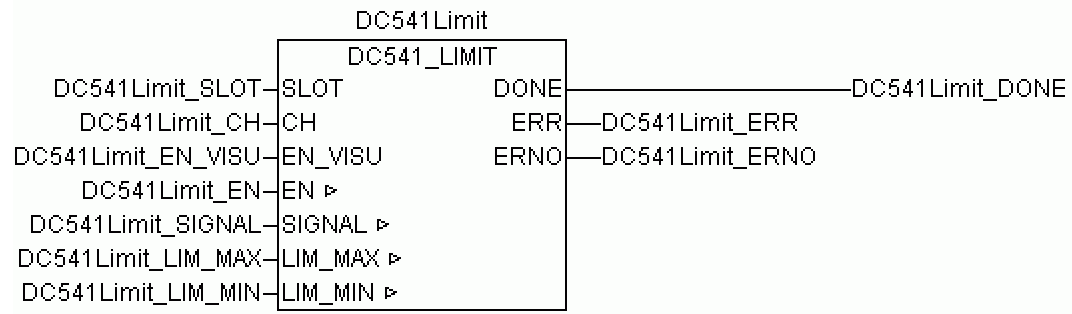
```
LD    DC541Io.ERNO
ST    DC541Io_ERNO

LD    DC541Io.IN0
ST    DC541Io_IN0

LD    DC541Io.IN1
ST    DC541Io_IN1

LD    DC541Io.IN2
ST    DC541Io_IN2

LD    DC541Io.IN3
ST    DC541Io_IN3

LD    DC541Io.IN4
ST    DC541Io_IN4

LD    DC541Io.IN5
ST    DC541Io_IN5

LD    DC541Io.IN6
ST    DC541Io_IN6

LD    DC541Io.IN7
ST    DC541Io_IN7
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541Io (EN   := DC541Io_EN,
         SLOT := DC541Io_SLOT,
         OUT0 := DC541Io_OUT0,
         OUT1 := DC541Io_OUT1,
         OUT2 := DC541Io_OUT2,
         OUT3 := DC541Io_OUT3,
         OUT4 := DC541Io_OUT4,
         OUT5 := DC541Io_OUT5,
         OUT6 := DC541Io_OUT6,
         OUT7 := DC541Io_OUT7);

DC541Io_DONE := DC541Io.DONE;
DC541Io_ERR  := DC541Io.ERR;
DC541Io_ERNO := DC541Io.ERNO;
DC541Io_IN0  := DC541Io.IN0;
DC541Io_IN1  := DC541Io.IN1;
DC541Io_IN2  := DC541Io.IN2;
DC541Io_IN3  := DC541Io.IN3;
DC541Io_IN4  := DC541Io.IN4;
DC541Io_IN5  := DC541Io.IN5;
DC541Io_IN6  := DC541Io.IN6;
DC541Io_IN7  := DC541Io.IN7;
```

DC541_LIMIT Limit value monitoring for the 32 bit counter



The block DC541_LIMIT is used for limit value monitoring of the 32 bit counter.

Block data

Available as of PLC runtime system:	V1.1.3	Remark:
Available as of DC541 firmware:	V1.1	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_LIMIT	Instance name
SLOT	Input	BYTE	Slot of the DC541
CH	Input	BYTE	Input selection C4...C7
EN_VISU	Input	BOOL	Enabling of control via the integrated visualization of block visuDC541_LIMIT
EN	Input/Output	BOOL	Enabling of the block processing
SIGNAL	Input/Output	BOOL	Signal level selection for output CH
LIM_MAX	Input/Output	DWORD	Counter end value at EN_LIM = TRUE
LIM_MIN	Input/Output	DWORD	Counter start value at EN_LIM = TRUE
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block DC541_LIMIT is used for limit value monitoring of the 32 bit counter. The block can be used to directly display various counting values of the 32 bit counter (DC541_32BIT_CNT) via binary outputs. Using the input SIGNAL you can determine whether the corresponding output is switched to FALSE or TRUE.

The time resolution of the block is < 100 µs, i.e. the result is increment accurate up to a frequency of 10 kHz.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If the lower limit value LIM_MIN is higher than the upper limit value LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

The device DC541 must be configured as counting device (counter mode) and channel C0 as 32 bit counter.

The block DC541_LIMIT has an integrated visualization visuDC541_LIMIT which can be used to control all block functions in parallel to the user program, if input EN_VISU is TRUE. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (see also "Integrated visualization of block DC541_FWD_CNT").

SLOT BYTE (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

CH BYTE (channel)

Input CH is used to select the channel for the limit value monitoring. Valid values are 4...7 for the outputs C4...C7. If an invalid value is specified at input CH or if the selected channel is not configured as Limit Channel 0 or if channel C0 is not configured as 32 bit counter, the block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

EN_VISU BOOL (enable input in visualization)

If input EN_VISU = TRUE, it is also possible to control the block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description (refer to "Integrated visualization of block DC541_LIMIT").

EN BOOL (enable)

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. If input EN = FALSE, the block is not processed and the pulses at input CH are lost.

When the block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

SIGNAL BOOL (signal)

Using the input SIGNAL you can determine whether the corresponding output CH is switched to TRUE or FALSE.

Counter value of the 32 bit counter DC541_32BIT_CNT / ACT_CNT	Output CH when SIGNAL = TRUE	Output CH when SIGNAL = FALSE
ACT_CNT < LIM_MIN	FALSE	TRUE
LIM_MIN <= ACT_CNT <= LIM_MAX	TRUE	FALSE
ACT_CNT > LIM_MAX	FALSE	TRUE
	CH ACT_CNT LIM_MIN LIM_MAX	CH ACT_CNT LIM_MIN LIM_MAX

If SIGNAL is TRUE, the output CH is TRUE, if the counter value of the 32 bit counter is within the range given by the limit values LIM_MIN and LIM_MAX. If the counter value is out of this range, output CH is FALSE.

If SIGNAL is FALSE, the output CH is FALSE, if the counter value of the 32 bit counter is within the range given by the limit values LIM_MIN and LIM_MAX. If the counter value is out of this range, output CH is TRUE.

LIM_MAX DWORD (limit maximum)

Input LIM_MAX is used to set the upper limit value for the limit value monitoring. The upper limit value LIM_MAX always has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

LIM_MIN DWORD (limit minimum)

Input LIM_MIN is used to set the lower limit value for the limit value monitoring. The upper limit value LIM_MAX always has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

```
CAL    DC541Limit(
       SLOT    := DC541Limit_SLOT,
       CH      := DC541Limit_CH,
       EN_VISU := DC541Limit_EN_VISU,
       EN      := DC541Limit_EN,
       SIGNAL  := DC541Limit_SIGNAL,
       LIM_MAX := DC541Limit_LIM_MAX,
       LIM_MIN := DC541Limit_LIM_MIN)

LD     DC541Limit.DONE
ST     DC541Limit_DONE

LD     DC541Limit.ERR
ST     DC541Limit_ERR

LD     DC541Limit.ERNO
ST     DC541Limit_ERNO
```

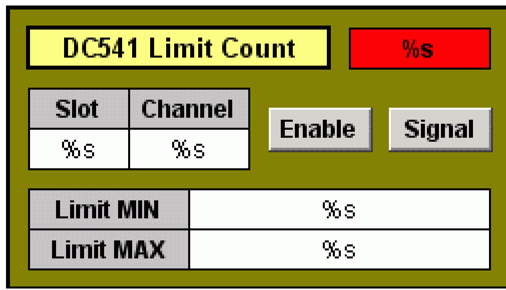
Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541Limit  (SLOT    := DC541Limit_SLOT,
             CH      := DC541Limit_CH,
             EN_VISU := DC541Limit_EN_VISU,
             EN      := DC541Limit_EN,
             SIGNAL  := DC541Limit_SIGNAL,
             LIM_MAX := DC541Limit_LIM_MAX,
             LIM_MIN := DC541Limit_LIM_MIN);

DC541Limit_DONE  := DC541Limit.DONE;
DC541Limit_ERR   := DC541Limit.ERR;
DC541Limit_ERNO  := DC541Limit.ERNO;
```

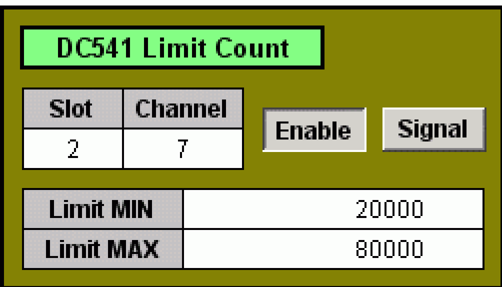
Integrated visualization of block DC541_LIMIT

The block DC541_LIMIT provides an integrated visualization visuDC541_LIMIT which can be included and instantiated in other visualizations. In offline mode of the Control Builder, the visualization looks as follows:



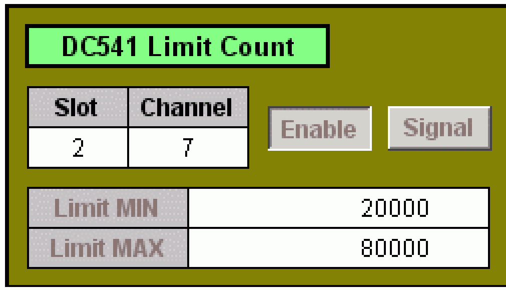
How to insert a visualization into a project and how to configure it is described at the beginning of the block library description (refer to Integrated visualization).

In online mode of the Control Builder, the visualization looks like in the following example, if EN_VISU is set to TRUE:

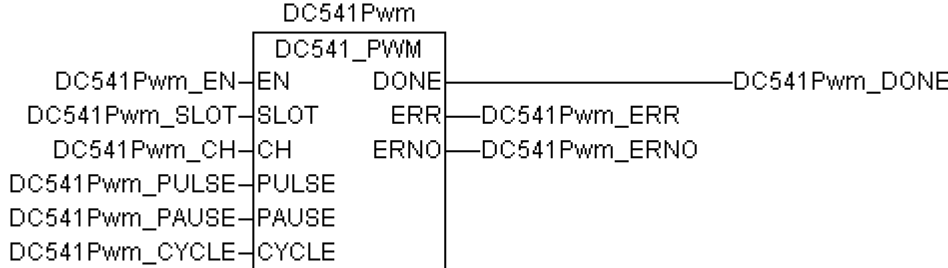


If an error occurred during block processing, the error number is displayed in the top right until EN becomes FALSE.

If input EN_VISU = FALSE, the inputs cannot be modified using the visualization. The corresponding control elements are then displayed in gray:



DC541_PWM Pulse-width modulator



The block DC541_PWM outputs a pulsed signal with an adjustable on-off ratio.

Block data

Available as of PLC runtime system:	V1.1.3	Remark:
Available as of DC541 firmware:	V1.1	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_PWM	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot of the DC541
CH	Input	BYTE	Output selection C0...C7
PULSE	Input	BYTE	Ratio of the TRUE signal
PAUSE	Input	BYTE	Ratio of the FALSE signal
CYCLE	Input	WORD	Minimum switching time [µs]
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block DC541_PWM outputs a pulsed signal with an adjustable on-off ratio. The on and off times are adjusted as 8 bit numbers.

The minimum switching time is specified at input CYCLE, i.e. if an output has been switched to FALSE or TRUE by the PWM, this output remains in this state for at least this time (CYCLE µs).

The minimum time specified at input CYCLE must not be smaller than the cycle time of the device DC541. Depending on its configuration, the cycle time of the DC541 can be 50, 100 or 200 µs. The cycle time can be polled using the block DC541_GET_CFG (output CYCLE) (refer to block DC541_GET_CFG/CYCLE).

Examples:

PULSE	PAUSE	CYCLE	Result (x = number of cycles of the DC541)
Cycle time of DC541 = 50 µs			
1	2	500	10 x TRUE / 20 x FALSE / 10 x TRUE / 20 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE
4	8	500	10 x TRUE / 20 x FALSE / 10 x TRUE / 20 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE (as in example 1, i.e. ratio 1 : 2)
3	2	3000	90 x TRUE / 60 x FALSE / 90 x TRUE / 60 x FALSE / … i.e. 4500 µs = TRUE and 3000 µs = FALSE
Cycle time of DC541 = 100 µs			
1	2	500	5 x TRUE / 10 x FALSE / 5 x TRUE / 10 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE
4	8	500	5 x TRUE / 10 x FALSE / 5 x TRUE / 10 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE (as in example 1, i.e. ratio 1 : 2)
3	2	3000	45 x TRUE / 30 x FALSE / 45 x TRUE / 30 x FALSE / … i.e. 4500 µs = TRUE and 3000 µs = FALSE
Cycle time of DC541 = 200 µs			
1	2	500	3 x TRUE / 6 x FALSE / 3 x TRUE / 6 x FALSE /... i.e. 600 µs = TRUE, 1200 µs = FALSE
4	8	500	3 x TRUE / 6 x FALSE / 3 x TRUE / 6 x FALSE / ... i.e. 600 µs = TRUE, 1200 µs = FALSE (as in example 1, i.e. ratio 1 : 2)
3	2	3000	22 x TRUE / 15 x FALSE / 23 x TRUE / 15 x FALSE / … i.e. first 4400 µs = TRUE and 3000 µs = FALSE and then 4600 µs = TRUE and 3000 µs = FALSE

The device DC541 must be configured as counting device (counter mode).

EN BOOL (enable)

Input EN has to be continuously set to TRUE.

When the block is called for the first time, input SLOT is checked for validity, the device configuration is checked for the correct configuration "counting mode" and the configuration of channel CH is checked whether it is configured as "PWM". If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO. The block outputs are updated as long as input EN = TRUE.

SLOT BYTE (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

CH BYTE (channel)

Input CH is used to select the output to be pulsed.

PULSE BYTE (pulse)

Input PULSE is used to specify the ratio for the TRUE signal (on time ratio).

PAUSE BYTE (pause)

Input PAUSE is used to specify the ratio for the FALSE signal (off time ratio).

CYCLE WORD (cycle)

Input CYCLE is used to specify the minimum switching time of the output in [μs]. The output remains in the TRUE or FALSE state for at least the time specified at CYCLE.

The minimum time specified at input CYCLE must not be smaller than the cycle time of the device DC541. Depending on its configuration, the cycle time of the DC541 can be 50, 100 or 200 μs. The cycle time can be polled using the block DC541_GET_CFG (output CYCLE) (refer to block DC541_GET_CFG/CYCLE).

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

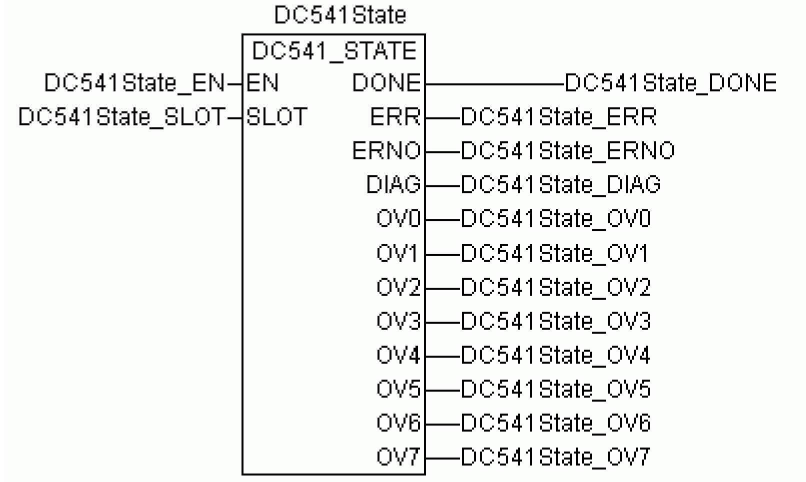
```
CAL    DC541Pwm(
       EN       := DC541Pwm_EN,
       SLOT     := DC541Pwm_SLOT,
       CH       := DC541Pwm_CH,
       PULSE    := DC541Pwm_PULSE,
       PAUSE    := DC541Pwm_PAUSE,
       CYCLE    := DC541Pwm_CYCLE)

LD     DC541Pwm.DONE
ST     DC541Pwm_DONE

LD     DC541Pwm.ERR
ST     DC541Pwm_ERR

LD     DC541Pwm.ERNO
ST     DC541Pwm_ERNO
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541Pwm(EN      := DC541Pwm_EN,
         SLOT    := DC541Pwm_SLOT,
         CH      := DC541Pwm_CH,
         PULSE   := DC541Pwm_PULSE,
         PAUSE   := DC541Pwm_PAUSE,
         CYCLE   := DC541Pwm_CYCLE);

DC541Pwm_DONE    := DC541Pwm.DONE;
DC541Pwm_ERR     := DC541Pwm.ERR;
DC541Pwm_ERNO    := DC541Pwm.ERNO;
```

DC541_STATE Status polling for the DC541



The block DC541_STATE is used to poll the status of the DC541.

Block data

Available as of PLC runtime system:	V1.1.3	Remark:
Available as of DC541 firmware:	V1.1	
Included in library:	DC541_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		DC541_STATE	Instance name
EN	Input	BOOL	Enabling of the block processing
SLOT	Input	BYTE	Slot of the DC541
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
DIAG	Output	BYTE	Diagnosis message of the DC541
OV0...OV7	Output	BOOL	Output status indication for the channels C0...C7

Description

The block DC541_STATE is used to poll the status of the device DC541. The information is displayed as long as input EN is TRUE. The displayed information includes the device diagnosis and the output status of the channels C0...C7.

EN BOOL (enable)

A FALSE -> TRUE edge at input EN activates the status polling. If the value at input SLOT is not valid, processing is aborted and a corresponding error is displayed at output ERR/ERNO. The block outputs are updated as long as input EN = TRUE. The block processing has been completed successfully, if output DONE changes to TRUE. During the processing of a request, state changes at input EN are recognized but not evaluated.

SLOT BYTE (slot)

Input SLOT is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during the processing of the block. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE. Encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

DIAG BYTE (diagnosis)

Output DIAG displays the diagnosis message of the DC541. If DIAG is 0, no diagnosis message is available. The following messages are possible:
16#01 = 1 -> Watchdog error

OV0...OV7 BOOL (overload C0...C7)

The outputs OV0..OV7 display the current output status of the channels C0...C7. If the respective bit is TRUE, the corresponding output cannot be switched e.g. due to an overload or a short circuit.

Function call in IL

```
CAL   DC541State (
      EN       := DC541State_EN,
      SLOT     := DC541State_SLOT)

LD    DC541State.DONE
ST    DC541State_DONE

LD    DC541State.ERR
ST    DC541State_ERR

LD    DC541State.ERNO
ST    DC541State_ERNO

LD    DC541State.DIAG
ST    DC541State_DIAG

LD    DC541State.OV0
ST    DC541State_OV0

LD    DC541State.OV1
ST    DC541State_OV1

LD    DC541State.OV2
ST    DC541State_OV2

LD    DC541State.OV3
ST    DC541State_OV3

LD    DC541State.OV4
ST    DC541State_OV4

LD    DC541State.OV5
ST    DC541State_OV5

LD    DC541State.OV6
ST    DC541State_OV6

LD    DC541State.OV7
ST    DC541State_OV7
```

Note: In IL, the function call has to be written in one line.

Function call in ST

```
DC541State(EN   := DC541State_EN,
          SLOT := DC541State_SLOT);

DC541State_DONE := DC541State.DONE;
DC541State_ERR  := DC541State.ERR;
DC541State_ERNO := DC541State.ERNO;
DC541State_DIAG := DC541State.DIAG;
DC541State_OV0  := DC541State.OV0;
DC541State_OV1  := DC541State.OV1;
DC541State_OV2  := DC541State.OV2;
DC541State_OV3  := DC541State.OV3;
DC541State_OV4  := DC541State.OV4;
DC541State_OV5  := DC541State.OV5;
DC541State_OV6  := DC541State.OV6;
DC541State_OV7  := DC541State.OV7;
```

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

Components of the DC541 library 2

D

DC541_32BIT_CNT 32 bit encoder 5

DC541_FREQ Time and frequency measurement 13

DC541_FREQ_FAST Fast time and frequency measurement 20

DC541_FREQ_OUT Frequency output 26

DC541_FWD_CNT 32 bit count-up counter 31

DC541_GET_CFG Output of DC541 configuration details 37

DC541_INT_IN Display of the interrupt initiating source 42

DC541_IO Reading/writing the inputs and outputs of the DC541 45

DC541_LIMIT Limit value monitoring for the 32 bit counter 48

DC541_PWM Pulse-width modulator 53

DC541_STATE Status polling for the DC541 57

G

Glossary 60

I

Integrated visualization of blocks contained in the DC541 library 3

O

Overview of blocks arranged according to their call names 3

P

Preconditions for the use of the library 2

S

Special characteristics of the DC541 library 2

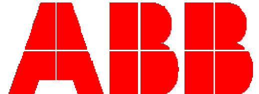
Software Description

AC500

Scalable PLC
for Individual Automation

Counter
Function Block Library

Counter



Contents

Counter Library 2

Preconditions for the use of the library 2

Special characteristics of the Counter library 2

Components of the Counter library 2

Overview of blocks arranged according to their call names 2

Integrated visualization of blocks contained in the Counter library 2

CNT_DC551 High-speed counter of DC551-CS31 5

CNT_IO High-speed counter of digital S500 I/O devices 13

Glossary 21

Index 23

Counter Library

Preconditions for the use of the library

Note:

The blocks contained in the counter library can only be executed in RUN mode of the PLC, but not in simulation mode.

The blocks of the Counter_AC500_V11.lib are available in AC500 control systems with runtime system version V1.1.7 or higher and S500 I/O devices with firmware version V1.3 or higher.

Special characteristics of the counter library

The counter library contains blocks that simplify the use of the high-speed counters of the S500 I/O devices on the AC500 CPU I/O bus and the DC551 on the CS31 bus respectively. The configuration of the high-speed counters is done in the PLC configuration. A detailed description of the AC500 PLC configuration can be found in the chapter "System Technology of the CPUs / PLC configuration".

The library Counter_AC500_V11.lib is not automatically included into a project. AC500_V11.lib is automatically included with the next compilation of the project.

Components of the counter library

The counter library contains the following function blocks:

Group:	
CNT_DC551	High-speed counter of DC551-CS31
CNT_IO	High-speed counter of digital S500 I/O devices

Overview of blocks arranged according to their call names

Character description:

FBhV … Function block with historical values

FBnohV … Function block without historical values

F … Function

VE name	Type	Function
CNT_DC551	FBhv	High-speed counter of DC551-CS31
CNT_IO	FBhv	High-speed counter of digital S500 I/O devices

Integrated visualization of blocks contained in the counter library

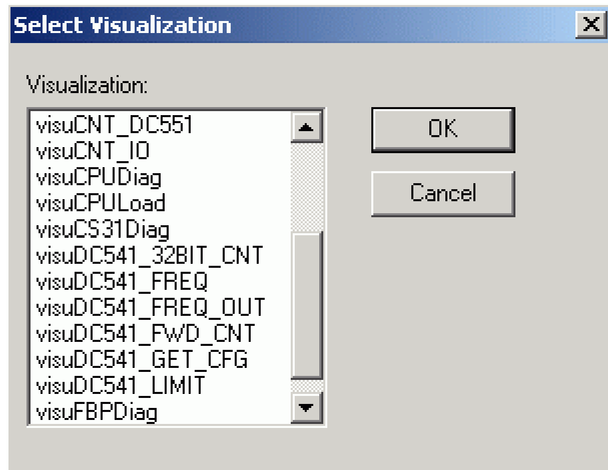
The following blocks of the library Counter_AC500_V11.LIB have an integrated visualization:

Block	Visualization name
CNT_DC551	visuCNT_DC551
CNT_IO	visuCNT_IO

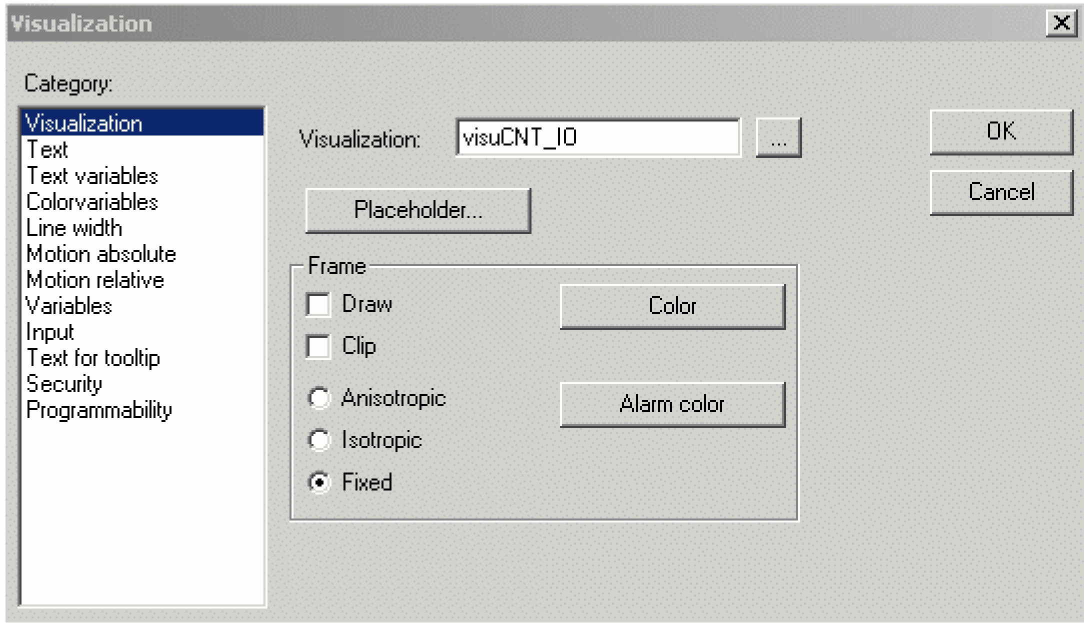
The visualization can be used to display the block outputs. If the input EN_VISU of a block is TRUE, it is also possible to control the inputs from the visualization. In order to allow the control of block inputs from the program as well as from the visualization, these inputs are declared as VAR_IN_OUT and therefore have to be provided with variables accordingly. These inputs must not be provided with direct constants.

Proceed as follows to integrate the visualization into a project:

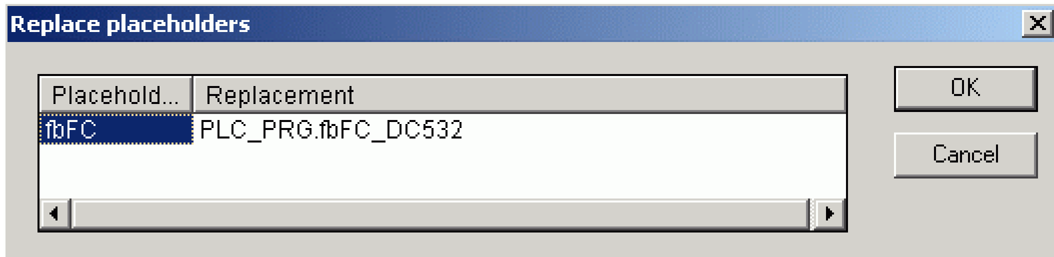
- Create a new visualization using Visualizations / Insert Object (e.g. visuTestCntIO).
- Insert a visualization using
- In the appearing dialog, select the corresponding visualization for the block:



- Then, the visualization integrated just now has to be configured. Highlight the visualization with a left mouse click. Then click the right mouse button and select the function "Configure..." from the context menu.
- The configuration of the visualization is done in the appearing dialog. It is recommended to set the frame to "Fixed". This way, the original width-to-height ratio and font size are kept.

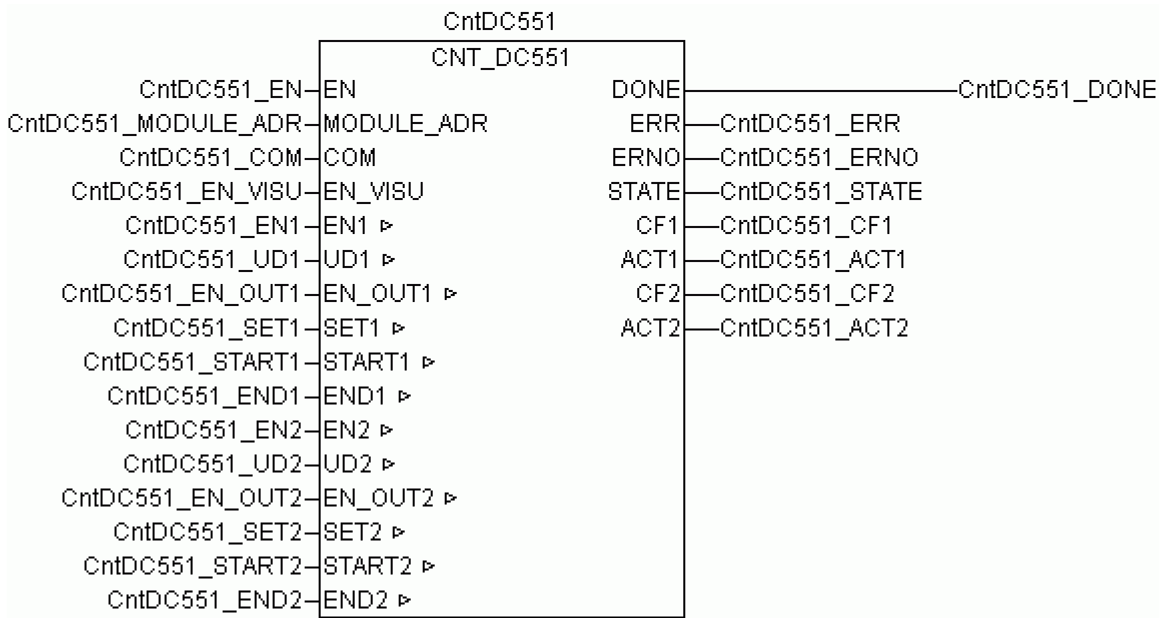


- Now the visualization has to be linked to the block instance. This is done in a dialog, opened after clicking the button "Placeholder".
- In the "Replacement" column of this dialog, the block instance can either be entered directly or selected by pressing <F2>.



- By clicking <OK> the dialogs are closed. After this, the inserted visualization has to be adapted to the correct size.

CNT_DC551 High-speed counter of DC551-CS31



The block CNT_DC551 is used to control the high-speed counter of the DC551.

Block data

Available as of PLC runtime system:	V1.0.2	Remark:
Available as of DC551 firmware:	V1.3	
Included in library:	Counter_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CNT_DC551	Instance name
EN	Input	BOOL	Enabling of the block processing
MODULE_ADR	Input	BYTE	Address of the DC551 (70...99)
COM	Input	BYTE	COMx of the AC500
EN_VISU	Input	BOOL	Enabling of control via the integrated visualization of block visuCNT_DC551
EN1	Input	BOOL	Enable counter 1
UD1	Input	BOOL	up = FALSE, down = TRUE, counter 1
EN_OUT1	Input	BOOL	Enable output for control via PLC program (only in mode 1 and mode 2)
SET1	Input	BOOL	Set input counter 1
START1	Input	DWORD	Start value counter 1
END1	Input	DWORD	End value counter 1
EN2	Input	BOOL	Enable counter 2 (only in modes with 2 counters)
UD2	Input	BOOL	up = FALSE, down = TRUE, counter 2

EN_OUT2	Input	BOOL	reserved (for enable output counter 2)
SET2	Input	BOOL	Set input counter 2
START2	Input	DWORD	Start value counter 2
END2	Input	DWORD	End value counter 2
DONE	Output	BOOL	Block processed completely
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
STATE	Output	WORD	Status display of the block
CF1	Output	BOOL	Overflow counter 1
ACT1	Output	DWORD	Current counter value, counter 1
CF2	Output	BOOL	Overflow counter 2
ACT2	Output	DWORD	Current counter value, counter 2

Description

The block CNT_DC551 is used to control the high-speed counter of the DC551.

The operating modes of the high-speed counter are described in detail in the chapter "High-speed counter" of the S500 I/O modules.

To activate the high-speed counter in the DC551, the address switch of the DC551 must be set to a value in the range between 70 and 99. Then, the actual module address is the set address minus 70, i.e. it is in a range between 0 and 29.

Data exchange between CPU and high-speed counter is done using input/output data. The following is required for the counter:

- 2 bytes digital inputs for status bytes 0 and 1
- 4 words analog inputs for 2 actual value double words of counters 1 and 2
- 2 bytes digital outputs for control bytes of counters 1 and 2
- 8 words analog outputs for 4 start/end value double words of counters 1 and 2

Thus, one DC551 with activated high-speed counter (without S500 expansion modules) occupies 2 CS31 software modules:

Module 1: digital module with digital inputs (3 bytes) and digital outputs (4 bytes)

Module 2: analog module with analog inputs (8 words) and analog outputs (4 words)

The address of the modules corresponds to the address set at the DC551 minus 70.

⚠ **Caution:** The high-speed counters of the S500 expansion modules are not available at the DC551. They are only available after connecting the modules to the I/O bus of the AC500 CPU.

The block CNT_DC551 has an integrated visualization visuCNT_DC551 that can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description.

EN BOOL (enable)

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. The block is not processed if input EN = FALSE.

When calling the block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

MODULE_ADR BYTE (module address)

At input MODULE_ADR, the address set at the DC551 is entered.

Valid values:

70...99 (corresponds to the CS31 module addresses 0...29)

COM BYTE (COM)

At input COM, the number of the serial interface is specified to which the DC551 is connected.

Valid values are 0 and 1 for COM1 of the AC500.

EN_VISU BOOL (enable visu)

If input EN_VISU = TRUE, it is also possible to control the block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description.

EN1 BOOL (enable counter 1)

If input EN1 = TRUE, pulse counting of counter 1 is enabled. If EN1 = FALSE, no pulse counting is performed and the pulses are lost.

Input EN1 corresponds to bit 1 in control byte 0.

UD1 BOOL (up/down counter 1)

At input UD1, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD1 = FALSE → counter 1 counts up
UD1 = TRUE → counter 1 counts down

If input SET1 = TRUE, the counter takes this value.

Input UD1 corresponds to bit 0 in control byte 0.

EN_OUT1 BOOL (enable output counter 1)

Input EN_OUT1 is used to select the output control mode for the operating modes with direct output activation (modes 1 and 2).

EN_OUT1 = FALSE:

If EN1 = TRUE, the pulses are counted. The output is activated, when running in an according operating mode with output activation.

If EN = FALSE, the block does not count and output control is performed according to the operating mode.

EN_OUT1 = TRUE:

If EN = TRUE, pulses are counted. However, in the corresponding operating modes with activation of the output, the output is **NOT** set/reset, i.e. no values are written to the output.

If EN = FALSE, the block does not perform counting and the output is likewise not activated.

If EN_OUT1 = TRUE, the output can be directly controlled by the PLC program.

Input EN_OUT1 corresponds to bit 3 in control byte 0.

SET1 BOOL (set counter 1)

If set input SET1 = TRUE, the counter takes the values from the inputs UD1, START1 and END1.

As long as input SET1 = TRUE, no pulses are counted because the counter is always overwritten by the start value START1.

Input SET1 corresponds to bit 2 in control byte 0.

START1 DWORD (start value 1)

At input START1, the start value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input START1 corresponds to the analog outputs 0 (bit 16..31) and 1 (bit 0..15) of the DC551.

END1 DWORD (end value 1)

At input END1, the end value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input END1 corresponds to the analog outputs 2 (bit 16..31) and 3 (bit 0..15) of the DC551.

UD2 BOOL (up/down counter 2)

At input UD2, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD2 = FALSE → counter 2 counts up
UD2 = TRUE → counter 2 counts down

If input SET2 = TRUE, the counter takes this value.

Input UD2 corresponds to bit 0 in control byte 1.

EN_OUT2 BOOL (enable output counter 2)

Input UD2 is reserved. A variable with the value FALSE has to be applied.

Input EN_OUT2 corresponds to bit 3 in control byte 1.

SET2 BOOL (set counter 2)

If set input SET2 = TRUE, the counter takes the values from the inputs UD2, START2 and END2.

As long as input SET2 = TRUE, no pulses are counted because the counter is always overwritten by the start value START2.

Input SET2 corresponds to bit 2 in control byte 1.

START2 DWORD (start value 2)

At input START2, the start value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input START2 corresponds to the analog outputs 4 (bit 16..31) and 5 (bit 0..15) of the DC551.

END2 DWORD (end value 2)

At input END2, the end value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input END2 corresponds to the analog outputs 6 (bit 16..31) and 7 (bit 0..15) of the DC551.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO indicates an error identification, if an invalid value is applied at one of the inputs. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

CF1 BOOL (carry flag 1)

If counter 1 has reached the programmed end value (input END1), output CF1 (end value reached) is set to TRUE and stored. When setting the counter (via input SET1), CF1 is set to FALSE.

Output CF1 corresponds to bit 0 in status byte 0.

ACT1 DWORD (actual value 1)

At output ACT1, the actual value = counter reading of counter 1 is output as double word.

Output ACT1 corresponds to the analog inputs 0 (bit 16..31) and 1 (bit 0..15).

CF2 BOOL (carry flag 2)

If counter 2 has reached the programmed end value (input END2), output CF2 (end value reached) is set to TRUE and stored. When setting the counter (via input SET2), CF2 is set to FALSE.

Output CF2 corresponds to bit 0 in status byte 1.

ACT2 DWORD (actual value 2)

At output ACT2, the actual value = counter reading of counter 2 is output as double word.

Output ACT2 corresponds to the analog inputs 2 (bit 16..31) and 3 (bit 0..15).

Function call in IL

```
CAL    CntDC551(
       EN            := CntDC551_EN,
       MODULE_ADR    := CntDC551_MODULE_ADR,
       COM           := CntDC551_COM,
       EN_VISU       := CntDC551_EN_VISU,
       EN1           := CntDC551_EN1,
       UD1           := CntDC551_UD1,
       EN_OUT1       := CntDC551_EN_OUT1,
       SET1          := CntDC551_SET1,
       START1        := CntDC551_START1,
       END1          := CntDC551_END1,
       EN2           := CntDC551_EN2,
       UD2           := CntDC551_UD2,
       EN_OUT2       := CntDC551_EN_OUT2,
       SET2          := CntDC551_SET2,
       START2        := CntDC551_START2,
       END2          := CntDC551_END2)

LD     CntDC551.DONE
ST     CntDC551_DONE
LD     CntDC551.ERR
ST     CntDC551_ERR
LD     CntDC551.ERNO
ST     CntDC551_ERNO
LD     CntDC551.CF1
ST     CntDC551_CF1
LD     CntDC551.ACT1
ST     CntDC551_ACT1
LD     CntDC551.CF2
ST     CntDC551_CF2
LD     CntDC551.ACT2
ST     CntDC551_ACT2
```

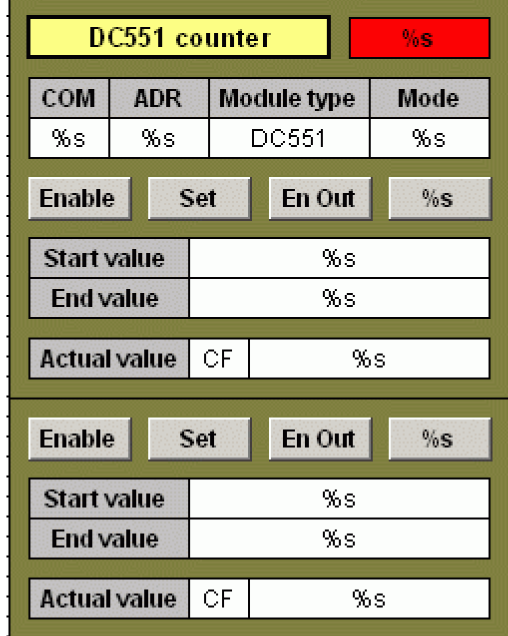
Note: In IL, the function call has to be written in one line.

Function call in ST

```
CntDC551 (EN          := CntDC551_EN
          MODULE_ADR := CntDC551_MODULE_ADR,
          COM         := CntDC551_COM,
          EN_VISU     := CntDC551_EN_VISU,
          EN1         := CntDC551_EN1,
          UD1         := CntDC551_UD1,
          EN_OUT1     := CntDC551_EN_OUT1,
          SET1        := CntDC551_SET1,
          START1      := CntDC551_START1,
          END1        := CntDC551_END1,
          EN2         := CntDC551_EN2,
          UD2         := CntDC551_UD2,
          EN_OUT2     := CntDC551_EN_OUT2,
          SET2        := CntDC551_SET2,
          START2      := CntDC551_START2,
          END2        := CntDC551_END2);

CntDC551_DONE      := CntDC551.DONE;
CntDC551_ERR       := CntDC551.ERR;
CntDC551_ERNO      := CntDC551.ERNO;
CntDC551_CF1       := CntDC551.CF1;
CntDC551_ACT1      := CntDC551.ACT1;
CntDC551_CF2       := CntDC551.CF2;
CntDC551_Act2      := CntDC551.ACT2;
```

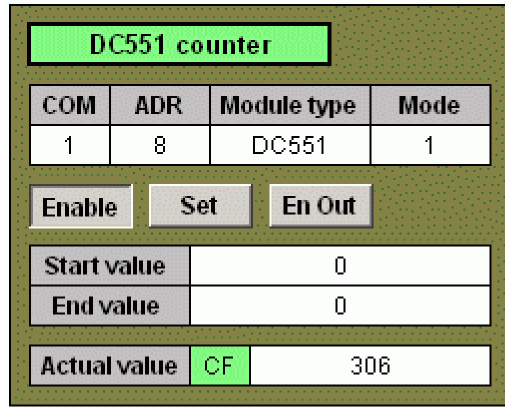
Integrated visualization of block CNT_DC551

The block CNT_DC551 provides an integrated visualization visuCNT_DC551 that can be included and instantiated in other visualizations. In offline mode of the Control Builder, the visualization looks as follows:



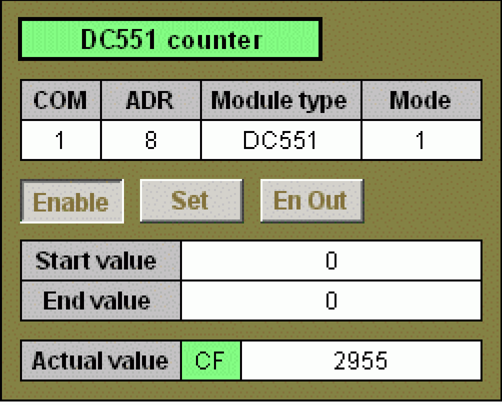
How to insert a visualization into a project and how to configure this, is described at the beginning of the block library description (refer to Integrated visualization).

In online mode of the Control Builder, the visualization looks as shown in the following example for operating mode 1:

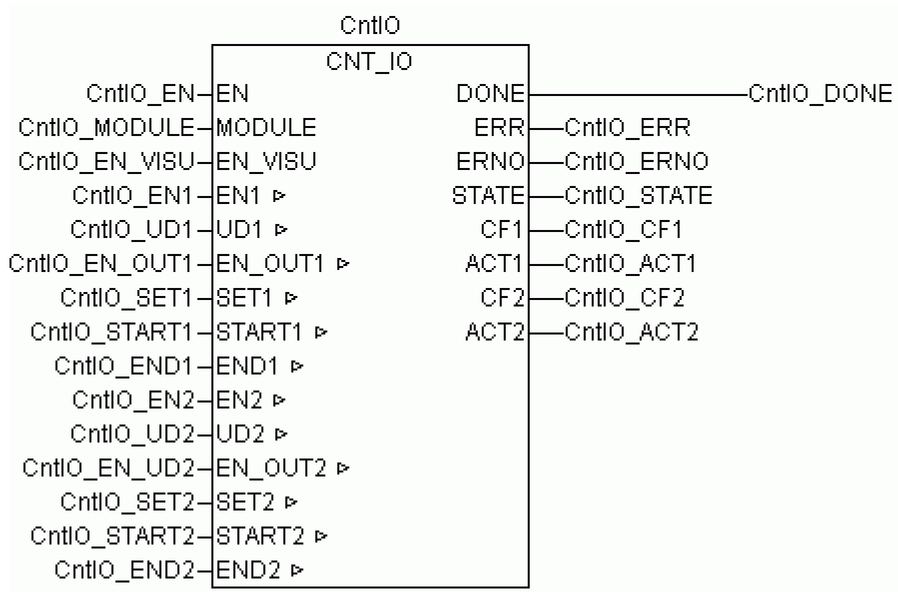


If an error occurred during block processing, the error number is displayed in the top right until EN becomes FALSE.

If input EN_VISU = FALSE, the inputs cannot be modified using the visualization. The corresponding control elements are then displayed in gray:


DC551 counter
COM
ADR
Module type
Mode
1
8
DC551
1
Enable
Set
En Out
Start value
0
End value
0
Actual value
CF
2955

CNT_IO High-speed counter of digital S500 I/O devices



The block CNT_IO is used to control the high-speed counter of the digital S500 I/O devices.

Block data

Available as of PLC runtime system:	V1.0.2	Remark:
Available as of S500 I/O device firmware:	V1.3	
Included in library:	Counter_AC500_V11.LIB	

Block type

Function block with historical values

Parameters

Instance		CNT_IO	Instance name
EN	Input	BOOL	Enabling of the block processing
MODULE	Input	BYTE	Module number (extension) on I/O bus 1...7
EN_VISU	Input	BOOL	Enabling of control via the integrated visualization of block visuCNT_IO
EN1	Input	BOOL	Enable counter 1
UD1	Input	BOOL	up = FALSE, down = TRUE, counter 1
EN_OUT1	Input	BOOL	Enable output for control via PLC program (only in mode 1 and mode 2)
SET1	Input	BOOL	Set input counter 1
START1	Input	DWORD	Start value counter 1
END1	Input	DWORD	End value counter 1
EN2	Input	BOOL	Enable counter 2 (only in modes with 2 counters)
UD2	Input	BOOL	up = FALSE, down = TRUE, counter 2
EN_OUT2	Input	BOOL	reserved (for enable output counter 2)

SET2	Input	BOOL	Set input counter 2
START2	Input	DWORD	Start value counter 2
END2	Input	DWORD	End value counter 2
DONE	Output	BOOL	Block processed completely
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number
STATE	Output	WORD	Status display of the block
CF1	Output	BOOL	Overflow counter 1
ACT1	Output	DWORD	Current counter value, counter 1
CF2	Output	BOOL	Overflow counter 2
ACT2	Output	DWORD	Current counter value, counter 2

Description

The block CNT_IO is used to control the high-speed counter of the digital S500 I/O modules.

The operating modes of the high-speed counter are described in detail in the chapter "High-speed counter" of the S500 I/O modules.

To activate the high-speed counter of a digital S500 I/O device, the paramater "High-speed counter" of the I/O device must be set to the desired counting mode in the control system configuration.

Data exchange between CPU and high-speed counter is done using input/output data. The following is required for the counter:

- 2 bytes digital inputs for status bytes 0 and 1
- 4 words analog inputs for 2 actual value double words of counters 1 and 2
- 2 bytes digital outputs for control bytes of counters 1 and 2
- 8 words analog outputs for 4 start/end value double words of counters 1 and 2

⚠ **Caution:** The high-speed counters of the digital S500 I/O devices are only available when connected to the I/O bus of the AC500 CPUs. If connected to the CS31 bus (with DC551) or another fieldbus (with DC505), no high-speed counters are available.

The block CNT_IO has an integrated visualization visuCNT_IO that can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description.

EN BOOL (enable)

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. The block is not processed if input EN = FALSE.

When calling the block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

MODULE BYTE (module)

At input MODULE, the module number of the digital S500 I/O device on the I/O bus of the AC500 CPU is specified. The first module with number 1 is the module directly right to the CPU.

Valid values: 1 .. 7

EN_VISU BOOL (<u>en</u>able <u>visu</u>)

If input EN_VISU = TRUE, it is also possible to control the block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed. A detailed functional description of the visualization and how to integrate it can be found at the end of this block description.

EN1 BOOL (<u>en</u>able counter <u>1</u>)

If input EN1 = TRUE, pulse counting of counter 1 is enabled. If EN1 = FALSE, no pulse counting is performed and the pulses are lost.

Input EN1 corresponds to bit 1 in control byte 0.

UD1 BOOL (<u>u</u>p/<u>d</u>own counter <u>1</u>)

At input UD1, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD1 = FALSE → counter 1 counts up
UD1 = TRUE → counter 1 counts down

If input SET1 = TRUE, the counter takes this value.

Input UD1 corresponds to bit 0 in control byte 0.

EN_OUT1 BOOL (<u>en</u>able <u>out</u>put counter <u>1</u>)

Input EN_OUT1 is used to select the output control mode for the operating modes with direct output activation (modes 1 and 2).

EN_OUT1 = FALSE:

If EN1 = TRUE, the pulses are counted. The output is activated, when running in an according operating mode with output activation.

If EN = FALSE, the block does not count and output control is performed according to the operating mode.

EN_OUT1 = TRUE:

If EN = TRUE, pulses are counted. However, in the corresponding operating modes with activation of the output, the output is **NOT** set/reset, i.e. no values are written to the output.

If EN = FALSE, the block does not perform counting and the output is likewise not activated.

If EN_OUT1 = TRUE, the output can be directly controlled by the PLC program.

Input EN_OUT1 corresponds to bit 3 in control byte 0.

SET1 BOOL (<u>set</u> counter <u>1</u>)

If set input SET1 = TRUE, the counter takes the values from the inputs UD1, START1 and END1.

As long as input SET1 = TRUE, no pulses are counted because the counter is always overwritten by the start value START1.

Input SET1 corresponds to bit 2 in control byte 0.

START1 DWORD (start value 1)

At input START1, the start value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input START1 corresponds to the analog outputs 0 (bit 16..31) and 1 (bit 0..15) of the DC551.

END1 DWORD (end value 1)

At input END1, the end value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input END1 corresponds to the analog outputs 2 (bit 16..31) and 3 (bit 0..15) of the DC551.

UD2 BOOL (up/down counter 2)

At input UD2, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD2 = FALSE → counter 2 counts up
UD2 = TRUE → counter 2 counts down

If input SET2 = TRUE, the counter takes this value.

Input UD2 corresponds to bit 0 in control byte 1.

EN_OUT2 BOOL (enable output counter 2)

Input EN_OUT2 is reserved. The value FALSE has to be applied.

Input EN_OUT2 corresponds to bit 3 in control byte 1.

SET2 BOOL (set counter 2)

If set input SET2 = TRUE, the counter takes the values from the inputs UD2, START2 and END2.

As long as input SET2 = TRUE, no pulses are counted because the counter is always overwritten by the start value START2.

Input SET2 corresponds to bit 2 in control byte 1.

START2 DWORD (start value 2)

At input START2, the start value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input START2 corresponds to the analog outputs 4 (bit 16..31) and 5 (bit 0..15) of the DC551.

END2 DWORD (end value 2)

At input END2, the end value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input END2 corresponds to the analog outputs 6 (bit 16..31) and 7 (bit 0..15) of the DC551.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO indicates an error identification, if an invalid value is applied at one of the inputs. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

CF1 BOOL (carry flag 1)

If counter 1 has reached the programmed end value (input END1), output CF1 (end value reached) is set to TRUE and stored. When setting the counter (via input SET1), CF1 is set to FALSE.

Output CF1 corresponds to bit 0 in status byte 0.

ACT1 DWORD (actual value 1)

At output ACT1, the actual value = counter reading of counter 1 is output as double word.

Output ACT1 corresponds to the analog inputs 0 (bit 16..31) and 1 (bit 0..15).

CF2 BOOL (carry flag 2)

If counter 2 has reached the programmed end value (input END2), output CF2 (end value reached) is set to TRUE and stored. When setting the counter (via input SET2), CF2 is set to FALSE.

Output CF2 corresponds to bit 0 in status byte 1.

ACT2 DWORD (actual value 2)

At output ACT2, the actual value = counter reading of counter 2 is output as double word.

Output ACT2 corresponds to the analog inputs 2 (bit 16..31) and 3 (bit 0..15).

Function call in IL

```
CAL    CntDC551(
       EN            := CntIO_EN,
       MODULE        := CntIO_MODULE,
       EN_VISU       := CntIO_EN_VISU,
       EN1           := CntIO_EN1,
       UD1           := CntIO_UD1,
       EN_OUT1       := CntIO_EN_OUT1,
       SET1          := CntIO_SET1,
       START1        := CntIO_START1,
       END1          := CntIO_END1,
       EN2           := CntIO_EN2,
       UD2           := CntIO_UD2,
       EN_OUT2       := CntIO_EN_OUT2,
       SET2          := CntIO_SET2,
       START2        := CntIO_START2,
       END2          := CntIO_END2)

LD     CntIO.DONE
ST     CntIO_DONE
LD     CntIO.ERR
ST     CntIO_ERR
LD     CntIO.ERNO
ST     CntIO_ERNO
LD     CntIO.CF1
ST     CntIO_CF1
LD     CntIO.ACT1
ST     CntIO_ACT1
LD     CntIO.CF2
ST     CntIO_CF2
LD     CntIO.ACT2
ST     CntIO_ACT2
```

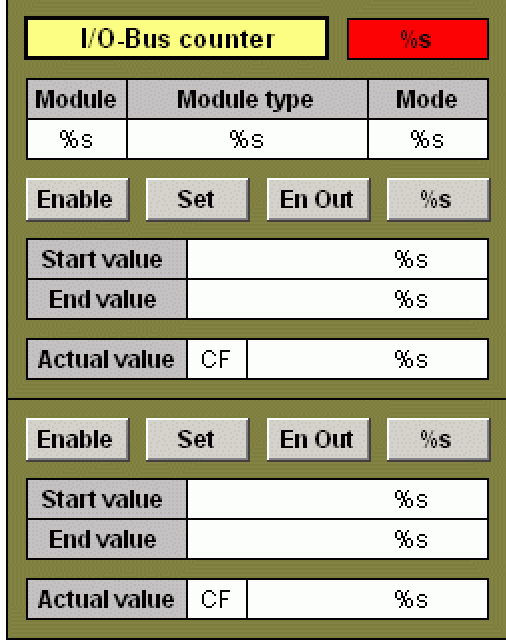
Note: In IL, the function call has to be written in one line.

Function call in ST

```
CntIO (EN         := CntIO_EN
        MODULE    := CntIO_MODULE,
        EN_VISU   := CntIO_EN_VISU,
        EN1       := CntIO_EN1,
        UD1       := CntIO_UD1,
        EN_OUT1   := CntIO_EN_OUT1,
        SET1      := CntIO_SET1,
        START1    := CntIO_START1,
        END1      := CntIO_END1,
        EN2       := CntIO_EN2,
        UD2       := CntIO_UD2,
        EN_OUT2   := CntIO_EN_OUT2,
        SET2      := CntIO_SET2,
        START2    := CntIO_START2,
        END2      := CntIO_END2);

CntIO_DONE        := CntIO.DONE;
CntIO_ERR         := CntIO.ERR;
CntIO_ERNO        := CntIO.ERNO;
CntIO_CF1         := CntIO.CF1;
CntIO_ACT1        := CntIO.ACT1;
CntIO_CF2         := CntIO.CF2;
CntIO_Act2        := CntIO.ACT2;
```

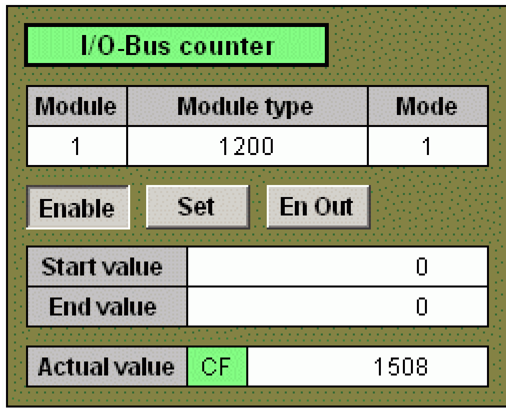
Integrated visualization of block CNT_IO

The block CNT_IO provides an integrated visualization visuCNT_IO that can be included and instantiated in other visualizations. In offline mode of the Control Builder, the visualization looks as follows:



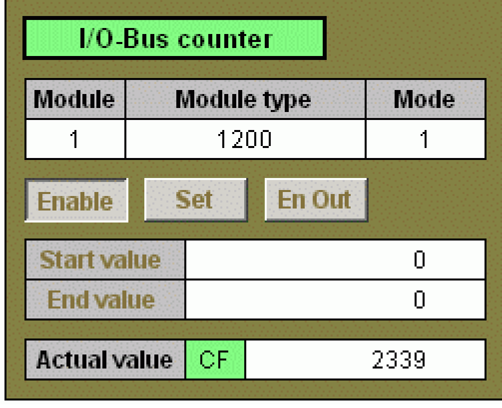
How to insert a visualization into a project and how to configure this, is described at the beginning of the block library description (refer to Integrated visualization).

In online mode of the Control Builder, the visualization looks as shown in the following example for operating mode 1:



If an error occurred during block processing, the error number is displayed in the top right until EN becomes FALSE.

If input EN_VISU = FALSE, the inputs cannot be modified using the visualization. The corresponding control elements are then displayed in gray:


I/O-Bus counter
Module
Module type
Mode
1
1200
1
Enable
Set
En Out
Start value
0
End value
0
Actual value
CF
2339

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

CNT_DC551 High-speed counter of DC551-CS31 5

CNT_IO High-speed counter of digital S500 I/O devices 13

Components of the counter library 2

G

Glossary 21

I

Integrated visualization of blocks contained in the counter library 2

O

Overview of blocks arranged according to their call names 2

P

Preconditions for the use of the library 2

S

Special characteristics of the counter library 2

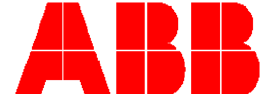
Software Description

AC500

Scalable PLC
for Individual Automation

Diagnosis
Function Block Library

Diagnosis



Contents

Diagnosis Library 2

Preconditions for the use of the library 2

Special characteristics of the Diagnosis library 2

Components of the Diagnosis library 2

Overview of blocks arranged according to their call names 3

Integrated visualization of blocks contained in the Diagnosis library 3

CPU_DIAG Reading the AC500 diagnosis buffer 7

CPU_LOAD Output of the CPU load 11

CS31_DIAG Diagnosis of the CS31 bus 15

FBP_DIAG Diagnosis of the FBP slave interface 21

Glossary 26

Index 28

Diagnosis Library

Preconditions for the use of the library

Note:

The blocks contained in the diagnosis library can only be executed in RUN mode of the PLC, but not in simulation mode. In simulation mode, the integrated visualization shows "Simulation".

Special characteristics of the diagnosis library

The blocks of the diagnosis library enable the direct access on the data of the AC500 diagnosis system.

The blocks are contained in the library Diag_AC500_V10.lib. The library is not loaded automatically into an AC500 project.

Components of the diagnosis library

Function blocks

The diagnosis library contains the following function blocks:

Group: CPU_Diagnosis	
CPU_DIAG	Reading the AC500 diagnosis buffer
CPU_LOAD	Output of the CPU capacity utilization

Group: CS31_Bus_Diagnosis	
CS31_DIAG	Diagnosis of the CS31 bus

Group: FBP_Diagnosis	
FBP_DIAG	Diagnosis of the FBP slave interface

Data types

The following data types are defined in the Diag_AC500_V10.lib:

Group: Type_CPU_Diagnosis	
AC500_Diag_Entry	Structure of an AC500 diagnosis entry
strCPU_LOAD	Structure of the CPU load

Group: Type_CS31_Diagnosis	
strCS31_DiagBus	Diagnosis structure of CS31 master
strCS31_DiagModule	Diagnosis structure of all modules
strCS31_DiagOneModule	Diagnosis structure of a slave at the CS31 bus

Group: Type_FBP_Diagnosis	
strFBP_Info	Diagnosis structure of the FBP slave interface
strFBP_ModuleInfo	Diagnosis structure of a module of the FBP slave interface
strFBP_Statistics	Diagnosis structure with statistic values of the FBP slave interface

Visualizations

The following visualizations are contained in the Diag_AC500_V10.lib:

Group:	
Visu_CPU_Diag	Visualization of the CPU diagnosis
Visu_CPU_Load	Visualization of the CPU load
Visu_CS31_Diag	Visualization of the CS31 diagnosis
Visu_FBP_Diag	Visualization of the FBP slave interface diagnosis

Global variables lists

Variables list: GL_AC500_Diagnosis		
Variable	**Type**	**Description**
CPU	CPU_LOAD	Instance of the block CPU load
diagCPU	CPU_DIAG	Instance of the block CPU diagnosis
diagCS31	CS31_DIAG	Instance of the block CS31 diagnosis
diagFBP	FBP_DIAG	Instance of the block FBP slave diagnosis

Variables list: GL_Diag_Constant		
Variable	**Type**	**Description**
wERNO_SIMULATION_MODE	WORD	16#50FF = 20735 dec, error number for simulation mode

Overview of blocks arranged according to their call names

Character description:

FBhV … Function block with historical values

FBnohV … Function block without historical values

F … Function

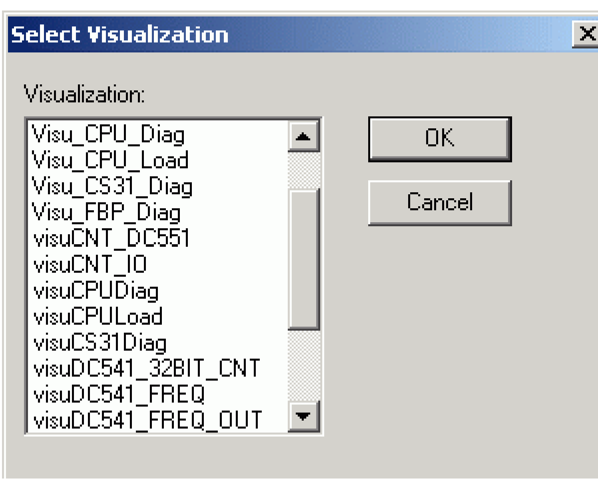
VE name	Type	Function
CPU_DIAG	FBhv	Reading the AC500 diagnosis buffer
CPU_LOAD	FBhv	Output of the CPU capacity utilization
CS31_DIAG	FBhv	Diagnosis of the CS31 bus
FBP_DIAG	FBhv	Diagnosis of the FBP slave interface

Integrated visualization of blocks contained in the diagnosis library

This chapter describes how to integrate the visualization of the blocks into an AC500 project.

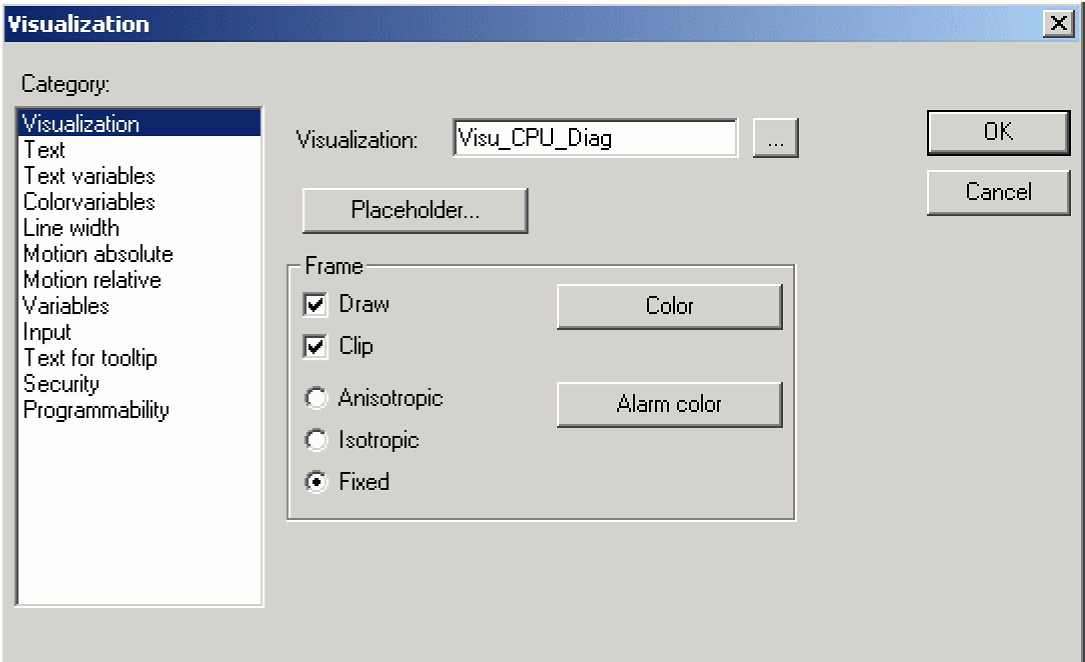
Proceed as follows to integrate the visualization into a project:

- Create a new visualization using Visualizations / Insert Object (e.g. visuCPUDiag).
- Insert a visualization using
- In the appearing dialog, select the corresponding visualization for the block:
  e. g. Visu_CPU_Diag



- Then, the visualization integrated just now has to be configured. Highlight the visualization with a left mouse click. Then click the right mouse button and select the function "Configure..." from the context menu.

- The configuration of the visualization is done in the appearing dialog. It is recommended to set the frame to "Fixed". This way, the original width-to-height ratio and font size are kept.

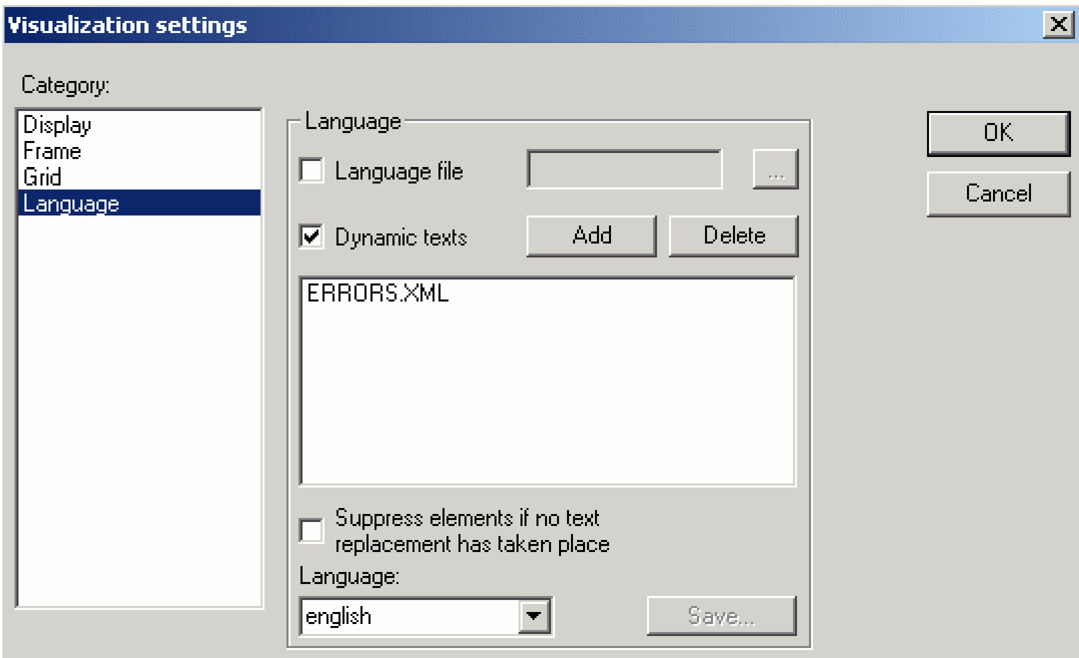


- By clicking on <OK> the dialogs are closed. After this, the inserted visualization has to be adapted to the correct size.

In the visualization of the block CPU_DIAG, the diagnosis messages shall be output as language-dependent plain text. Proceed as follows to include the corresponding text file "Errors.xml":

- Switch to the visualization editor.

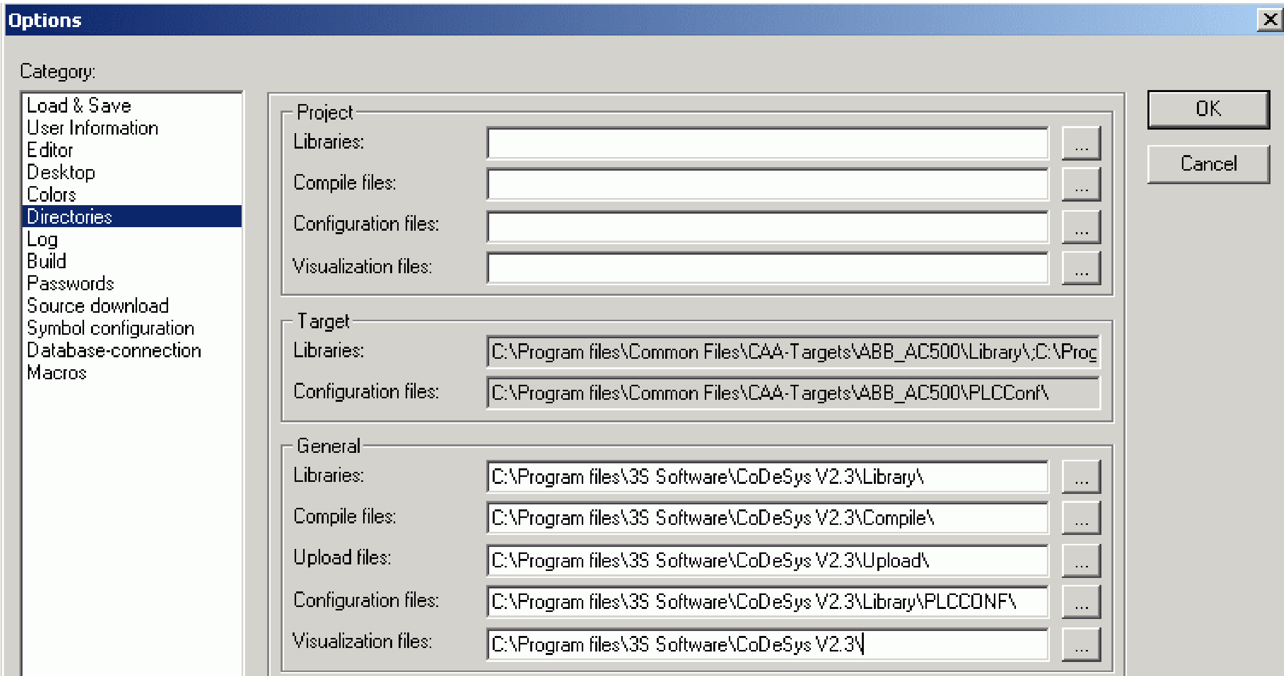
- Select the menu item "Extras"/"Settings". In the appearing dialog, select the category "Language", check "Dynamic texts" and enter "ERRORS.XML" as file name. Now, the desired language can be selected in the field "Language".



- Then, the path has to be entered in order to find the language file "ERRORS.XML". Select "Project"/"Options"/"Directories", go to "General"/"Visualization files" and enter the path CoDeSys.exe. In case of a default installation, this is:

  C:\Program Files\3S Software\CoDeSys V2.3 or C:\Programme\3S Software\CoDeSys V2.3.

  During installation of the PS501, the file ERRORS.XML is copied to this directory.



The following little example illustrates how to integrate all 4 diagnosis blocks into a project and how to operate them via a central visualization.

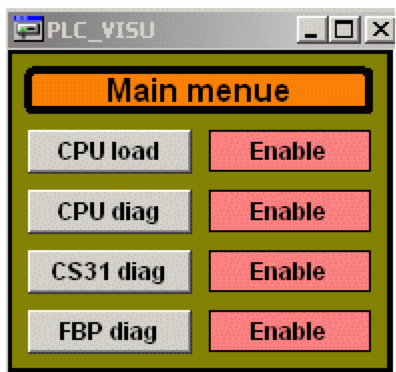
In the same way as described for the block CPU_DIAG, the visualizations for the blocks CPU_LOAD (visuCPULoad), CS31_DIAG (visuCS31Diag) and FBP_DIAG (visuFBPDiag) have to be created.

By means of the visualization PLC_VISU, the individual visualizations can be launched and enabled or disabled. For that purpose, the program "PLC_Diagnosis()" is included into the AC500 project:

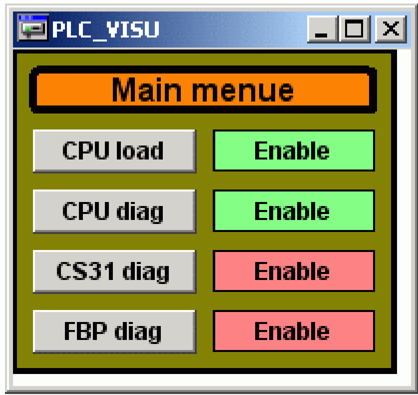
The program contains the following declarations and calls:

```
PROGRAM PLC_Diagnosis

VAR
	bCPUEnable : BOOL := TRUE;
	bCS31Enable : BOOL := FALSE;
	bFBPEnable : BOOL := FALSE;
	bCPUDiagEnable : BOOL := TRUE;
END_VAR

(* Call AC500 diagnosis *)
CPU(EN := bCPUEnable);
diagCPU(EN := bCPUDiagEnable);
diagCS31(EN := bCS31Enable, COM := 1);
diagFBP(EN := bFBPEnable);
```

In offline mode, the related visualization PLC_VISU looks as follows:



After enabling the CPU diagnosis and CPU load, PLC_VISU is displayed as follows in online mode:

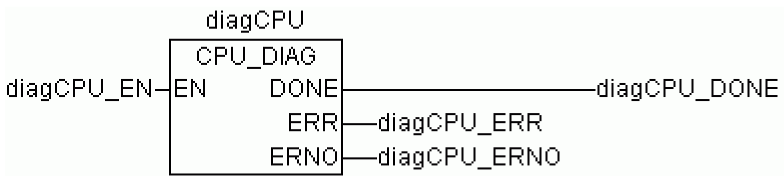


Using the buttons on the left side, the corresponding visualization is called. Clicking the "Enable" buttons enables the processing of the related blocks.

The program "PLC_Diagnosis()" and the visualization PLC_VISU are contained in the export file **AC500_PLC_Diagnosis.exp** and can be inserted into an AC500 project via Project/Import. The export file is located on the PS501 CD-ROM under:

..\CD_AC500\Examples\PLC_Diagnosis.

CPU_DIAG Reading the AC500 diagnosis buffer



The block CPU_DIAG is used to the read the entries contained in the AC500 diagnosis system.

Block data

Available as of PLC runtime system:	V1.0.2	Remark:
Included in library:	Diag_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		CPU_DIAG	Instance name
EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

Using the block CPU_DIAG, the entries of the AC500 diagnosis system can be read and displayed using the integrated visualization as plain text.

The instance diagCPU of the block CPU_DIAG is declared in the global variables list **GL_AC500_Diagnosis**. The integrated visualization of the block accesses the variables of this instance.

In order to include the block into an AC500 project, it is only necessary to call the block instance. Example in ST:

diagCPU(EN := TRUE);

By means of the visualization **Visu_CPU_Diag**, navigation within the diagnosis buffer is possible and upcoming diagnosis entries can be acknowledged.

The steps how to use the visualization are described at the end of this block description.

EN BOOL (enable)

EN = TRUE enables the processing of the block. With a FALSE -> TRUE edge at input EN, the last five entries are read from the diagnosis buffer of the CPU.

DONE BOOL (done)

Output DONE indicates the state of job processing. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

If the block is called in simulation mode, error 16#50FF = 20735 is reported.

Function call in IL

Note: The instance **diagCPU** of the block CPU_DIAG is declared in the global variables list **GL_AC500_Diagnosis** of the library **Diag_AC500_V10.LIB** and has therefore just to be called.

```
CAL   diagCPU(EN := diagCPU_EN)

LD    diagCPU.DONE
ST    diagCPU_DONE
LD    diagCPU.ERR
ST    diagCPU_ERR
LD    diagCPU.ERNO
ST    diagCPU_ERNO
```

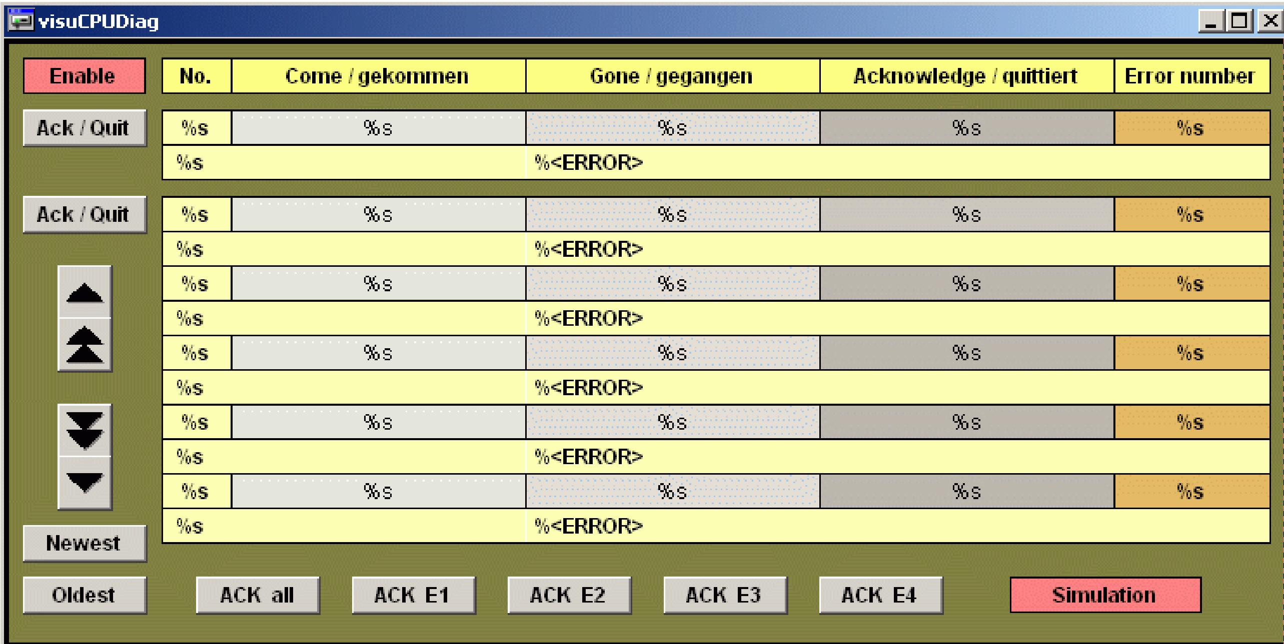
Note: In IL, the function call has to be written in one line.

Function call in ST

```
diagCPU(EN := diagCPU_EN);

diagCPU_DONE := diagCPU.DONE;
diagCPU_ERR  := diagCPU.ERR;
diagCPU_ERNO := diagCPU.ERNO;

or:

diagCPU(EN := TRUE);
```

Integrated visualization of block CPU_DIAG

The visualization Visu_CPU_Diag is part of the block CPU_DIAG:



The steps how to integrate and call the visualization in an AC500 project are explained in detail at the beginning of the library description.

This section describes how to operate the visualization.

The following is output for each diagnosis entry:

- current error number "No."
- time stamps for errror Come / Gone / Acknowledge
- "Error number"
- short text and error text taken from "ERRORS.XML"

In the upper area, the last diagnosis entry is displayed. This entry can be acknowledged by pressing the button <Ack / Quit>.

In the middle, the last 5 diagnosis entries are displayed after a FALSE/TRUE edge. The buttons on the left allow to navigate within the diagnosis buffer:

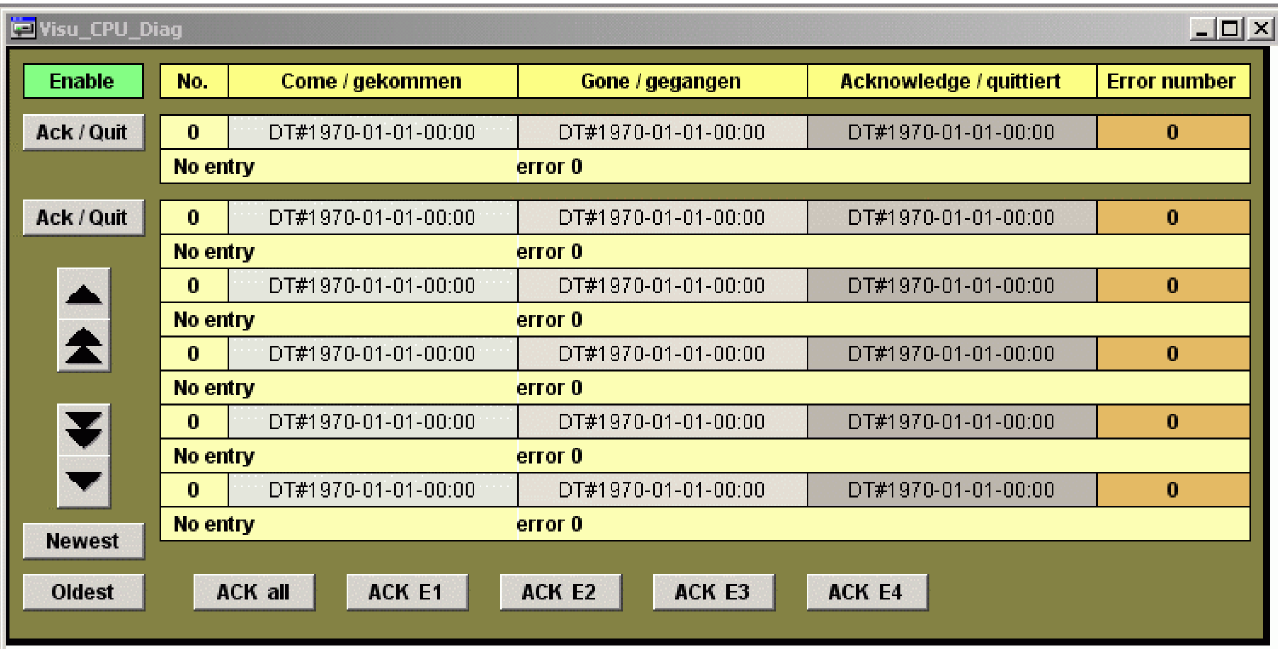
- <Ack / Quit>	- acknowledges the diagnosis entry displayed next to the button
- arrow up	- next entry
- arrow down	- previous entry
- double arrow up	- 5 entries forward
- double arrow down	- 5 entries back
- Newest	- 5 entries back starting with the latest entry
- Oldest	- 5 entries forward starting with the oldest entry

Using the buttons below the diagnosis entries, errors can be acknowledged:

- <ACK all>	- acknowledges all errors
- <ACK E1>	- acknowledges E1 errors (fatal errors)
- <ACK E2>	- acknowledges E2 errors (serious errors)
- <ACK E3>	- acknowledges E3 errors (light errors)
- <ACK E4>	- acknowledges E4 errors (warnings)

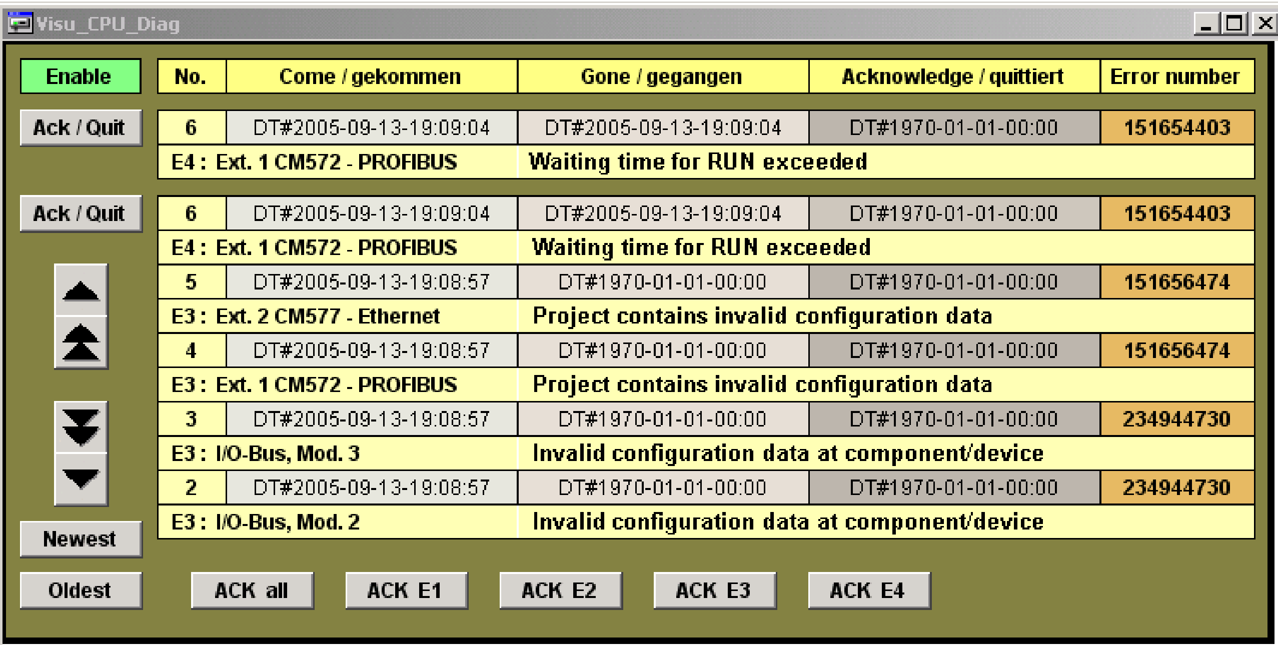
When acknowlegding errors, the blocks DIAG_ACK and DIAG_ACK_ALL of the library SysInt_AC500_V10.LIB are called.

If no diagnosis entries exist, the visualization in online mode looks as follows:

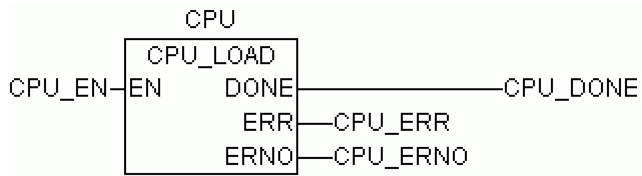


Remark: The PLC real-time clock (RTC) is not set in the example.

If errors exist, the visualization could, for example, appear as follows:



CPU_LOAD Output of the CPU capacity utilization



The block CPU_LOAD is used to output the CPU capacitiy utilization.

Block data

Available as of PLC runtime system:	V1.0.2	Remark:
Included in library:	Diag_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		CPU_LOAD	Instance name
EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block CPU_LOAD outputs the CPU capacitiy utilization in [%].

The following is displayed:

current load
minimum load
maximum load
average (avg) load

The instance **CPU** of the block CPU_LOAD is declared in the global variables list **GL_AC500_Diagnosis**. The integrated visualization of the block accesses the variables of this instance.

In order to include the block into an AC500 project, it is only necessary to call the block instance. Example in ST:

CPU(EN := TRUE);

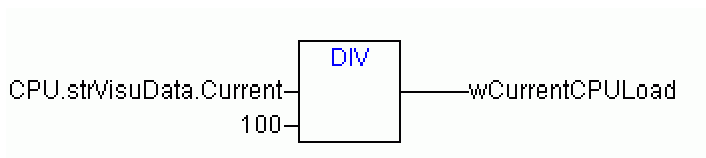
The values are entered as 0.01% values into the internal structure **strVisuData of the type strCPU_LOAD** in the instance CPU of the block CPU_LOAD. To obtain the values in [%], they have to be divided by 100. The structure is composed as follows:

```
TYPE strCPU_LOAD :
STRUCT
    Current     : WORD;     (* current value *)
    Avg         : WORD;     (* average value *)
    Minimum     : WORD;     (* minimum value *)
    Maximum     : WORD;     (* maximum value *)
    Counter     : DWORD;    (* counter *)
    Conf        : WORD;     (* bit 0 = 1 (1) --> Update CPU_load *)
                            (* bit 1 = 1 (2) --> Heap check *)
END_STRUCT
END_TYPE
```

If the user program should have access to these values, this is done as follows:

CPU.strVisuData.Variable.

In the following example, the value of the current CPU load is read and assigned to the variable wCurrentCPULoad:



Using the visualization **Visu_CPU_Load**, the values are represented graphically. The steps how to use the visualization are described at the end of this block description.

The same values can also be polled using the PLC browser command "cpuload".

EN BOOL (enable)

EN = TRUE enables the processing of the block and updates the values at each call.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

Note: The instance **CPU** of the block CPU_LOAD is declared in the global variables list **GL_AC500_Diagnosis** of the library **Diag_AC500_V10.LIB** and has therefore just to be called.

```
CAL    CPU(EN := CPU_EN)

LD     CPU.DONE
ST     CPU_DONE
LD     CPU.ERR
ST     CPU_ERR
LD     CPU.ERNO
ST     CPU_ERNO
```

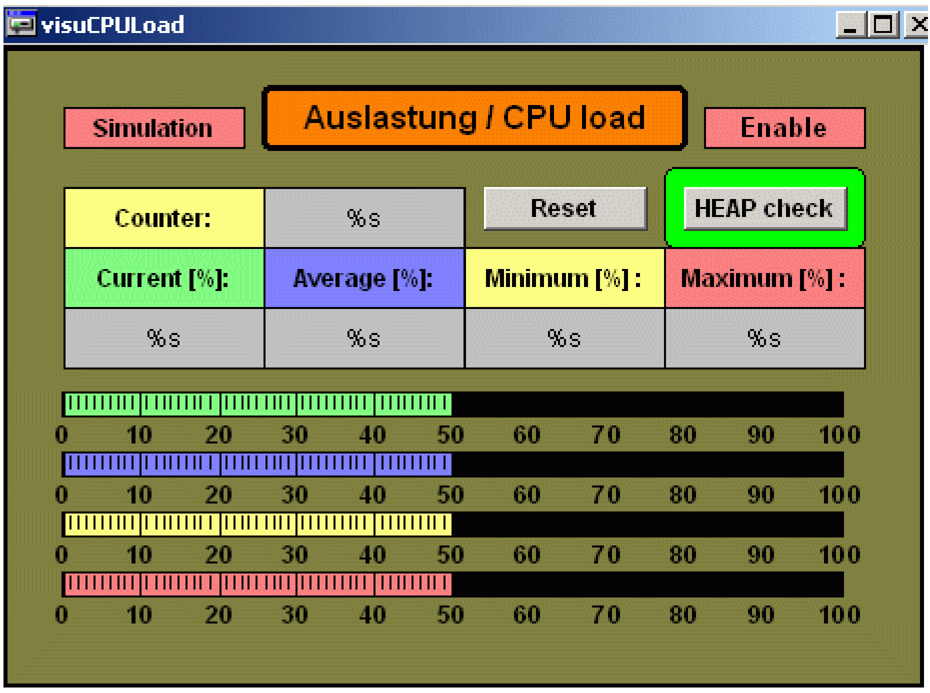
Note: In IL, the function call has to be written in one line.

Function call in ST

```
CPU(EN := CPU_EN);

CPU_DONE := CPU.DONE;
CPU_ERR  := CPU.ERR;
CPU_ERNO := CPU.ERNO;

or:

CPU(EN := TRUE);
```

Integrated visualization of block CPU_LOAD

The visualization **Visu_CPU_Load** is part of the block CPU_LOAD:

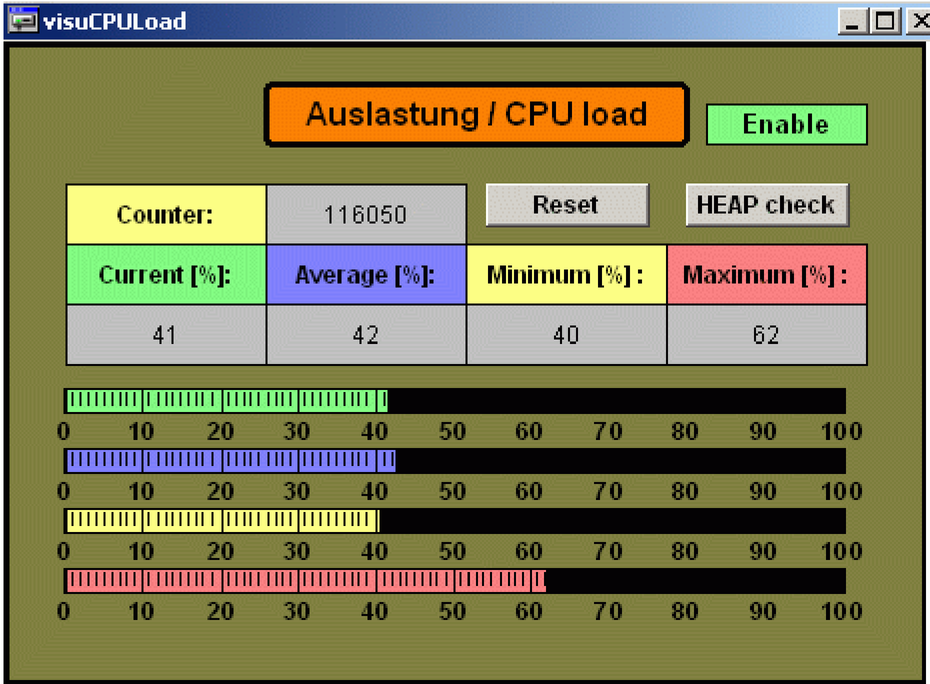


The steps how to integrate and call the visualization in an AC500 project are explained in detail at the beginning of the library description.

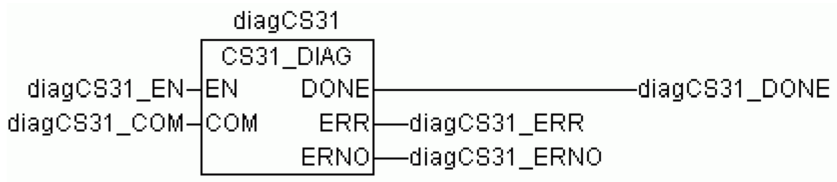
If input EN = TRUE, the CPU load values in [%] are cyclically output in a numerical and graphical (bar display) representation. Pressing the button <Reset> resets the values.

The button <HEAP check> checks the internal HEAP memory of the CPU. While the HEAP check is activated, no load values are calculated and output.

In online mode, the visualization appears, for example, as follows:



CS31_DIAG Diagnosis of the CS31 bus



The block CS31_DIAG is used for the diagnosis of the CS31 bus.

Block data

Available as of PLC runtime system:	V1.0.2	Remark:
Included in library:	Diag_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		CS31_DIAG	Instance name
EN	Input	BOOL	Enabling of the block processing
COM	Input	BYTE	Number of the serial interface of the CS31 master: 0 or 1 for COM1
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block CS31_DIAG provides the diagnosis data of the CS31 bus to the user.

The instance **diagCS31** of the block CS31_DIAG is declared in the global variables list **GL_AC500_Diagnosis**. The integrated visualization of the block accesses the variables of this instance.

In order to include the block into an AC500 project, it is only necessary to call the block instance. Example in ST:

CPU(EN := TRUE, COM := 1);

The read values are written into internal structures of the instance **diagCS31**.

Structure	TYPE	Assignment
strVisuData1	strCS31_DiagBus	Diagnosis data of the CS31 master
strVisuData	strCS31_DiagModule	Diagnosis data of all 31 CS31 software modules ARRAY[0..31] OF strCS31_DiagOneModule
CS31Mod	strCS31_DiagOneModule	Diagnosis data of one CS31 software module

The structure **strCS31_DiagBus** is composed as follows:

```
TYPE strCS31_DiagBus :
(* STRUCT info about CS31 bus *)
STRUCT
    iCS31BusState :          DINT;      (* Bit 0=1 waiting for slave
                                           Bit 1=1 initialization
                                           Bit 4=1 I/O data exchange
                                           := 19 all modules at bus *)
    iNumberModule :          DINT;      (* current no. of CS31 software modules on the CS31 bus *)
    iMaxNumberModule :       DINT;      (* max. number of modules at CS31 bus since power ON *)
    iErrorSum :              DINT;      (* sum error counter *)
    ulCycleCount :           DWORD;     (* counter of CS31 bus cycles *)
    byStateDiag :            BYTE;      (* general bus diagnosis: 0=ok, 1=no module on the bus *)
END_STRUCT
END_TYPE
```

The structure **strCS31_DiagOneModule** is composed as follows:

```
TYPE strCS31_DiagOneModule :
STRUCT
    dwScheduleCycle :        DWORD;     (* internal time *)
    byPhysicalAdress :       BYTE;      (* module address x 2 *)
    byCommState :            BYTE;      (* communication status:
                                               00 = OK
                                               Bit 0=1 (1) module disconnected again
                                               Bit 1=1 (2) module not available on bus
                                               Bit 2=1 (4) unconfigured module on bus
                                               Bit 3=1 (8) difference in I/O range
                                               Bit 4=1 (16) E4 &ndash; error message
                                               Bit 5=1 (32)
                                               Bit 6=1 (64) *)
    bySlaveType :            BYTE;      (* Slave type:
                                               00 = digital input
                                               01 = analog input
                                               02 = digital output
                                               03 = analog output
                                               04 = digital input/output
                                               05 = analog input/output
                                               06 = special module
    byInputBytes :           BYTE;      (* number of reported input bytes *)
    byOutputBytes :          BYTE;      (* number of reported output bytes *)
    byInputBytesConf :       BYTE;      (* number of input bytes in configuration *)
    byOutputBytesConf :      BYTE;      (* number of output bytes in configuration *)
    byModuleConf :           BYTE;      (* 1-PLC config, 3- ignore module *)
    byStateDiag :            BYTE;      (* internal flag *)
    byd1 :                   BYTE;      (* internal flag *)
    usErrorCounter :         WORD;      (* single-error counter *)
    byDiagErr :              BYTE;      (* internal error number *)
    byDiagChannnel :         BYTE;      (* internal error channel *)
    byd2 :                   BYTE;      (* internal flag *)
    byd3 :                   BYTE;      (* internal flag *)
    pConfig :                DWORD;     (* internal flag *)
    END_STRUCT
    END_TYPE
```

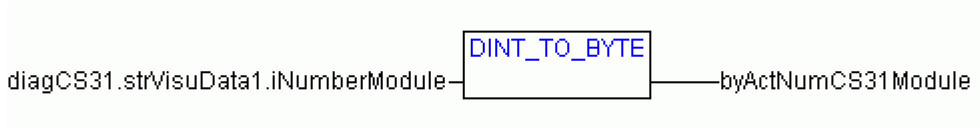
The structure strCS31_DiagModule contains an ARRAY[0..31], i.e. with 32 entries of the structure strCS31_DiagOneModule.

This way, the structures **strVisuData1** and **strVisuData** provide access to all diagnosis data of the CS31 bus.

If the user program should have access to these values, this is done as follows:

diagCS31.strVisuData1.Variable.

In the following example, the current number of CS31 modules is read and assigned to the variable byActNumCS31Module:



The most important data are available in the visualization Visu_CS31_Diag of the block. The steps how to use the visualization are described at the end of this block description.

EN BOOL (enable)

EN = TRUE enables the processing of the block and updates the values at each call.

COM BYTE (com)

At the COM input, the number of the serial interface of the CS31 master is specified.

Valid values: 0 and 1 for COM1

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

Note: The instance **diagCS31** of the block CS31_DIAG is declared in the global variables list **GL_AC500_Diagnosis** of the library **Diag_AC500_V10.LIB** and has therefore just to be called.

```
CAL    diagCS31(EN := diagCS31_EN, COM := diagCS31_COM)

LD     diagCS31.DONE
ST     diagCS31_DONE
LD     diagCS31.ERR
ST     diagCS31_ERR
LD     diagCS31.ERNO
ST     diagCS31_ERNO
```

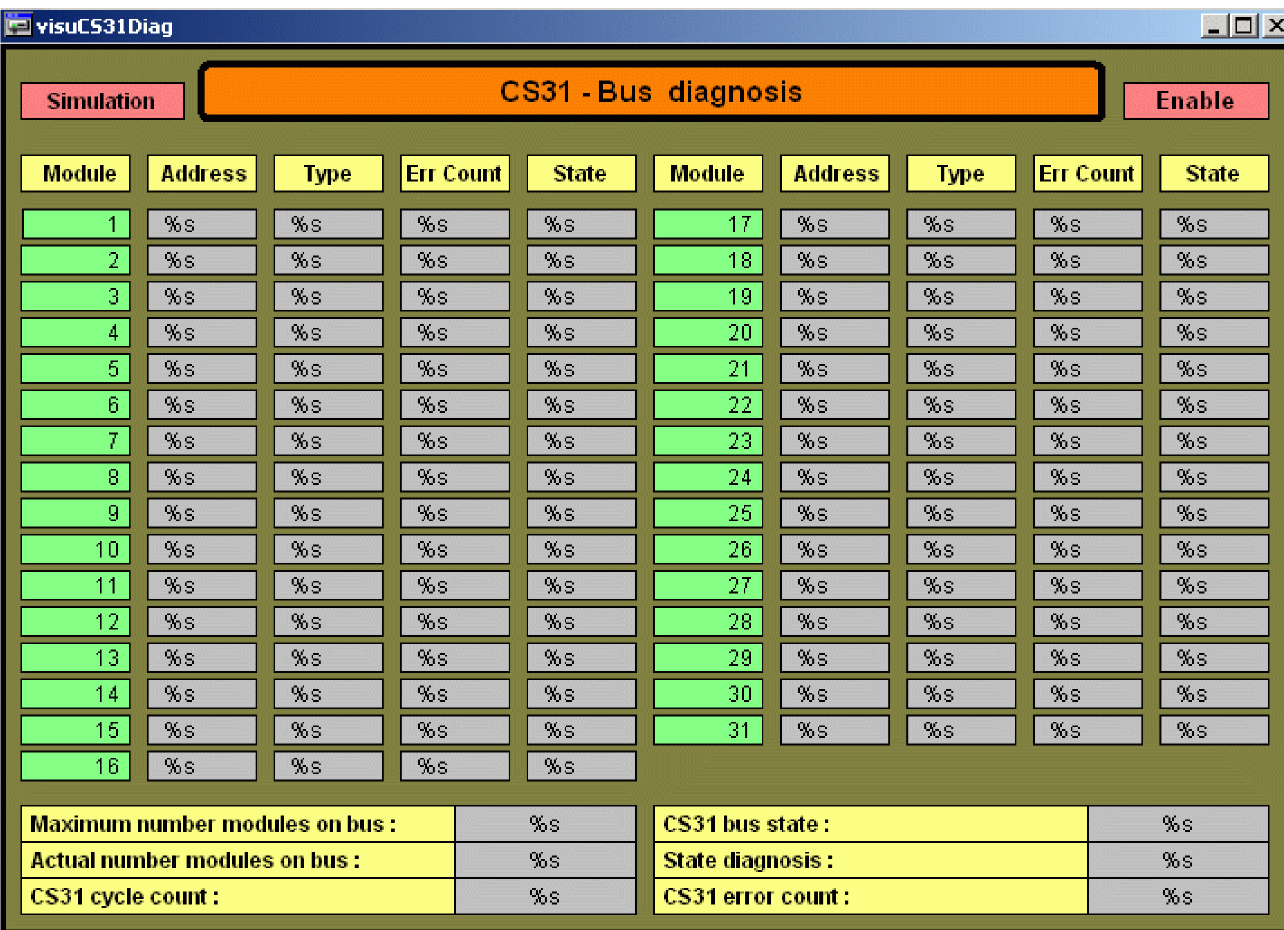
Note: In IL, the function call has to be written in one line.

Function call in ST

```
diagCS31(EN := diagCS31_EN, COM := diagCS31_COM);

diagCS31_DONE := diagCS31.DONE;
diagCS31_ERR  := diagCS31.ERR;
diagCS31_ERNO := diagCS31.ERNO;

or:

diagCS31(EN := TRUE, COM := 1);
```

Integrated visualization of block CS31_DIAG

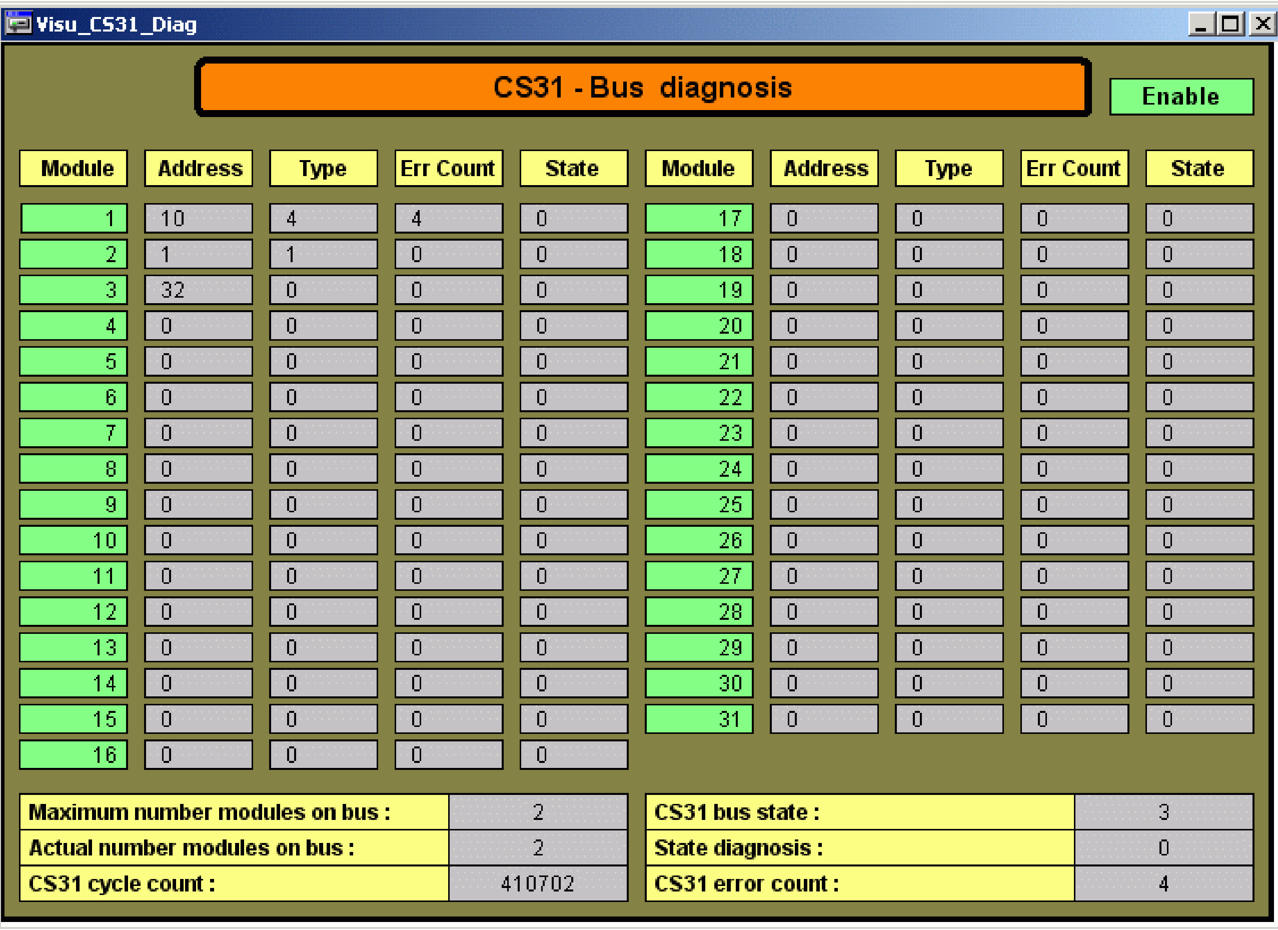
The visualization **Visu_CS31_Diag** is part of the block CS31_DIAG:



The steps how to integrate and call the visualization in an AC500 project are explained in detail at the beginning of the library description.

If input EN = TRUE, the values for the CS31 diagnosis are read cyclically and entered into the structures.

In online mode, the visualization appears, for example, as follows:



Visu_CS31_Diag

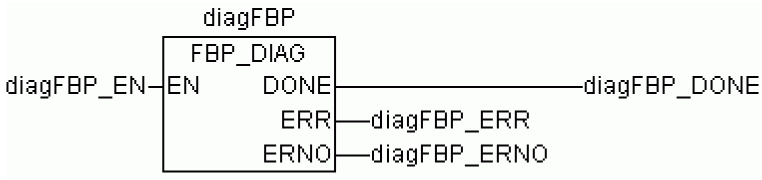
CS31 - Bus diagnosis

Enable

Module	Address	Type	Err Count	State	Module	Address	Type	Err Count	State
1	10	4	4	0	17	0	0	0	0
2	1	1	0	0	18	0	0	0	0
3	32	0	0	0	19	0	0	0	0
4	0	0	0	0	20	0	0	0	0
5	0	0	0	0	21	0	0	0	0
6	0	0	0	0	22	0	0	0	0
7	0	0	0	0	23	0	0	0	0
8	0	0	0	0	24	0	0	0	0
9	0	0	0	0	25	0	0	0	0
10	0	0	0	0	26	0	0	0	0
11	0	0	0	0	27	0	0	0	0
12	0	0	0	0	28	0	0	0	0
13	0	0	0	0	29	0	0	0	0
14	0	0	0	0	30	0	0	0	0
15	0	0	0	0	31	0	0	0	0
16	0	0	0	0					

Maximum number modules on bus :	2	CS31 bus state :	3
Actual number modules on bus :	2	State diagnosis :	0
CS31 cycle count :	410702	CS31 error count :	4

FBP_DIAG Diagnosis of the FBP slave interface



The block FBP_DIAG outputs the diagnosis data of the FBP slave interface.

Block data

Available as of PLC runtime system:	V1.0.2	Remark:
Included in library:	Diag_AC500_V10.LIB	

Block type

Function block with historical values

Parameters

Instance		FBP_DIAG	Instance name
EN	Input	BOOL	Enabling of the block processing
DONE	Output	BOOL	Completion of the block processing
ERR	Output	BOOL	Error message of the block
ERNO	Output	WORD	Error number

Description

The block FBP_DIAG is used to output the diagnosis data of the FBP slave interface.

The instance **diagFBP** of the block DIAG_FBP is declared in the global variables list **GL_AC500_Diagnosis**. The integrated visualization of the block accesses the variables of this instance.

In order to include the block into an AC500 project, it is only necessary to call the block instance. Example in ST:

diagFBP(EN := TRUE);

The read values are written into internal structures of the instance **diagFBP**.

Structure	TYPE	Assignment
strVisuData	strFBP_Info	Diagnosis data of the FBP slave interface
strVisuData1	strFBP_Statistics	Statistic data of the FBP slave interface

The structure **strFBP_Info** is composed as follows:

```
TYPE strFBP_Info :
STRUCT
    pbyBinInputs :              POINTER TO BYTE;    (* internal variable *)
    pbyBinOutputs :             POINTER TO BYTE;    (* internal variable *)
    pbyAnaInputs :              POINTER TO BYTE;    (* internal variable *)
    pbyAnaOutputs :             POINTER TO BYTE;    (* internal variable *)
    pParameter :                POINTER TO BYTE;    (* internal variable *)
    apsModule :                 ARRAY[1..8] OF      (* submodule info *)
                                POINTER TO
                                strFBP_ModuleInfo;
    usNumModules :              WORD;               (* number of modules *)
    usSizeParameters :          WORD;               (* parameter length in bytes *)
    usNumParameters :           WORD;               (* number of parameters *)
    usNumBinaryInputs :         WORD;               (* number of digital input bytes *)
    usSizeBinaryInputs :        WORD;               (* length of digital input bytes *)
    usNumBinaryOutputs :        WORD;               (* number of digital output bytes *)
    usSizeBinaryOutputs :       WORD;               (* length of digital output bytes *)
    usNumAnalogueInputs :       WORD;               (* number of analog input words *)
    usSizeAnalogueInputs :      WORD;               (* length of analog inputs in bytes *)
    usNumAnalogueOutputs :      WORD;               (* number of analog output words *)
    usSizeAnalogueOutputs :     WORD;               (* length of analog outputs in bytes *)
    ulBaudRate :                DWORD;              (* current baud rate *)
    byBusAddress :              BYTE;               (* bus address reported to FBP *)
    bySelectedTelegramType :    BYTE;               (* telegram type (acc. to FBP spec) *)
    bySelectedBaudRate :        BYTE;               (* set baud rate (acc. to FBP spec) *)
    byPrmFormat :               BYTE;               (* parameter format (acc. to FBP spec) *)
END_STRUCT
END_TYPE
```

The structure **strFBP_Statistics** is composed as follows:

```
TYPE strFBP_Statistics :
STRUCT
    ulNumInitRequestsSent :   DWORD;   (* no. of Inits sent to FBP since configuration *)
    ulNumRec :                DWORD;   (* no. of telegrams received from FBP since
                                       config *)
    ulNumSend :               DWORD;   (* no. of telegrams sent to FBP since config *)
    ulNumRecErrors :          DWORD;   (* number of faulty telegrams since config *)
    ulNumTimeouts :           DWORD;   (* number of timeouts since configuration *)
    ulNumChecksumErrors :     DWORD;   (* number of telegrams with checksum error *)
    ulNumInvalidRecs :        DWORD;   (* number of wrong/unknown telegrams *)
END_STRUCT
END_TYPE
```

If the user program should have access to these values, this is done as follows:

diagFBP.strVisuData.Variable.

In the following example, the FPB slave address is read and assigned to the variable byFBPAddress:

diagFBP.strVisuData.byBusAddress---------------byFBPAddress

Using the visualization **Visu_FBP_Diag**, the values are represented graphically. The steps how to use the visualization are described at the end of this block description.

EN BOOL (enable)

EN = TRUE enables the processing of the block and updates the values at each call.

DONE BOOL (done)

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR BOOL (error)

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO WORD (error number)

Output ERNO provides an error identifier if an invalid value has been applied to an input or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table "Error messages of the block libraries".

Function call in IL

Note: The instance **diagFBP** of the block FBP_DIAG is declared in the global variables list **GL_AC500_Diagnosis** of the library **Diag_AC500_V10.LIB** and has therefore just to be called.

```
CAL    diagFBP(EN := CPU_EN)

LD     diagFBP.DONE
ST     diagFBP_DONE
LD     diagFBP.ERR
ST     diagFBP_ERR
LD     diagFBP.ERNO
ST     diagFBP_ERNO
```

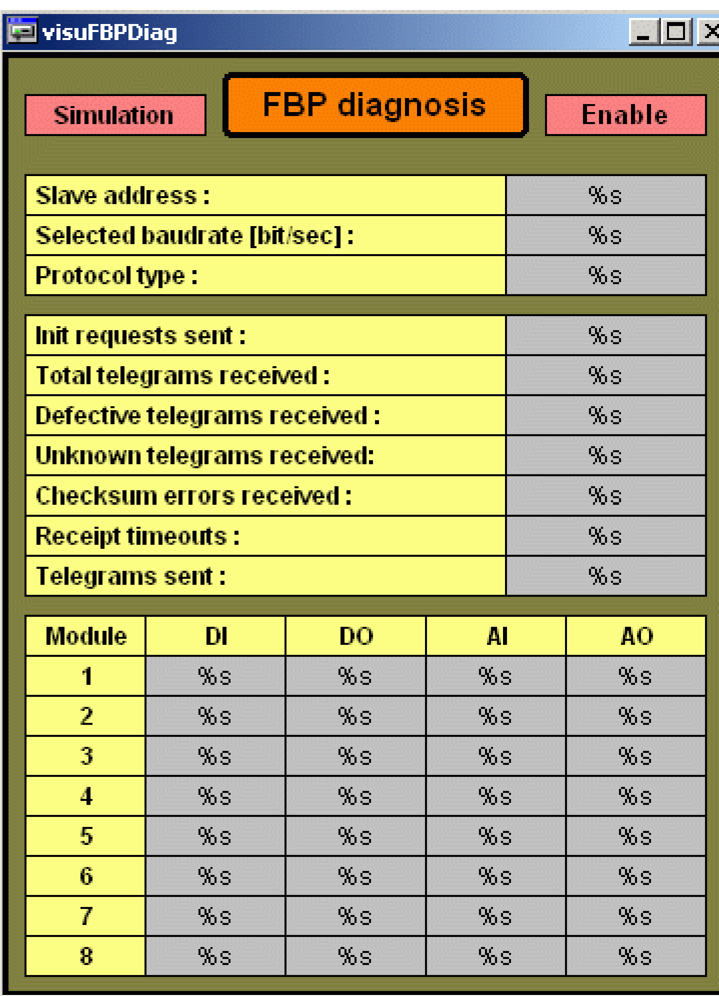
Note: In IL, the function call has to be written in one line.

Function call in ST

```
diagFBP(EN := diagFBP_EN);

diagFBP_DONE := diagFBP.DONE;
diagFBP_ERR  := diagFBP.ERR;
diagFBP_ERNO := diagFBP.ERNO;

or:

diagFBP(EN := TRUE);
```

Integrated visualization of block FBP_DIAG

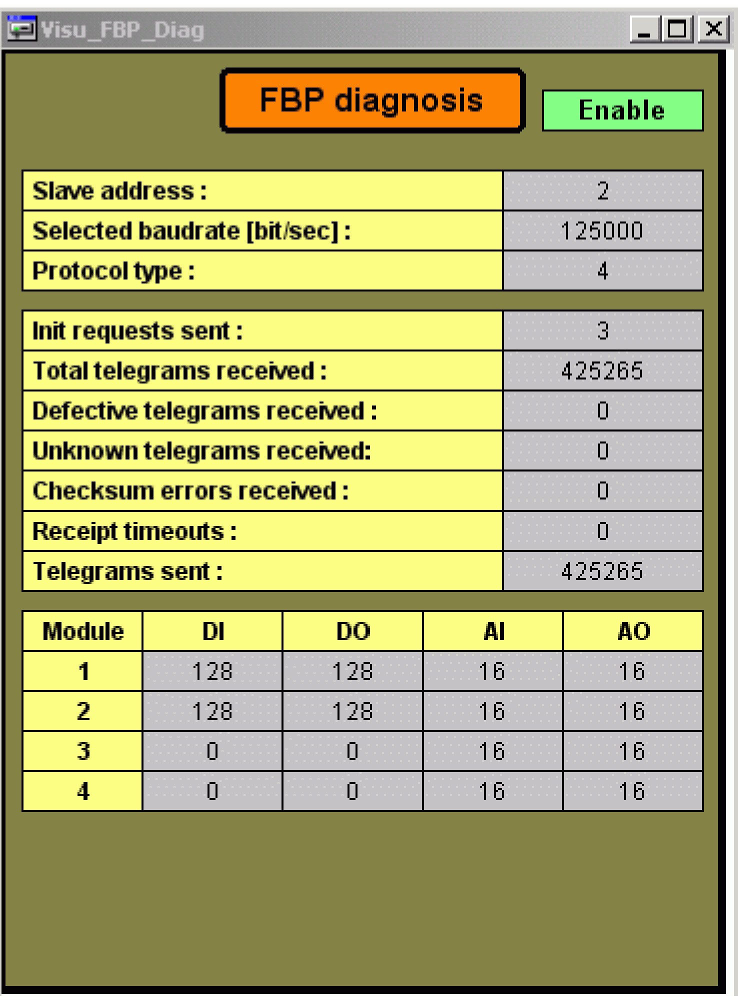
The visualization **Visu_FBP_Diag** is part of the block FBP_DIAG:



The steps how to integrate and call the visualization in an AC500 project are explained in detail at the beginning of the library description.

If input EN = TRUE, the values for the FBP diagnosis are read cyclically and entered into the structures.

In online mode, the visualization appears, for example, as follows:



The visualization displays the number of modules that are connected to the FBP slave interface in the PLC configuration.

Glossary

BOOL

Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE

BYTE belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types.

The different numerical types are responsible for a different numerical range. For integer data types the following range limits are valid:

Type	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on.

For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function blocks must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block.

As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated.

In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blocks can have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory):

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory):

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

For function blocks without historical values only one instance has to be defined for the FB type. This instance can be used for several calls of the FB (also with different I/O values).

Index

C

CPU_DIAG Reading the AC500 diagnosis buffer 7

CPU_LOAD Output of the CPU load 11

CS31_DIAG Diagnosis of the CS31 bus 15

Components of the diagnosis library 2

F

FBP_DIAG Diagnosis of the FBP slave interface 21

G

Glossary 26

I

Integrated visualization of blocks contained in the diagnosis library 3

O

Overview of blocks arranged according to their call names 3

P

Preconditions for the use of the library 2

S

Special characteristics of the diagnosis library 2

Manual No.: 2CDC 125 021 M0201

ABB STOTZ-KONTAKT GmbH
Eppelheimer Straße 82 69123 Heidelberg, Germany
Postfach 10 16 80 69006 Heidelberg, Germany
Telephone (06221) 701-0
Telefax (06221) 701-240
Internet http://www.abb.de/stotz-kontakt
E-Mail desst.helpline@de.abb.com