CAN / CANopen Extension for IEC 61131-3

User Manual

Version 7.0

Edition May 2011

Document No.: L-1008e_7

SYS TEC electronic GmbH Am Windrad 2 D-08468 Heinsdorfergrund
Telefon: +49 (3765) 38600-0 Telefax: +49 (3765) 38600-4100
Web: http://www.systec-electronic.com Mail: info@systec-electronic.com

Status/Changes

Status: Released

Date/Version	Section	Change	By
2007/04/02 6.0	2, 3 and 5	Sections 2, 3 and 5 have been revised completely	R. Sieber
2011/05/25 7.0	4.4.5, 4.4.6, 6	New Sections: 4.4.5 (CAN_SDO_READ_BIN), 4.4.6 (CAN_SDO_WRITE_BIN) and 6 (CAN Layer 2 FB)	R. Sieber

Product names used in this manual which are also registered trademarks have not been marked additionally. The missing © mark does not imply that the trade name is unregistered. Nor is it possible to determine the existence of any patents or protection of inventions on the basis of the names used.

The information in this manual has been carefully checked and is believed to be accurate. However, it is expressly stated that SYS TEC electronic GmbH does not assume warranty or legal responsibility or any liability for consequential damages which result from the use or contents of this user manual. The information contained in this manual can be changed without prior notice. Therefore, SYS TEC electronic GmbH shall not accept any obligation.

Furthermore, it is expressly stated that SYS TEC electronic GmbH does not assume warranty or legal responsibility or any liability for consequential damages which result from incorrect use of the hardware or software. The layout or design of the hardware can also be changed without prior notice. Therefore, SYS TEC electronic GmbH shall not accept any obligation.

Inform yourselves:

Contact	**Direct**	**Your local distributor**
Address:	SYS TEC electronic GmbH Am Windrad 2 D-08468 Heinsdorfergrund GERMANY	Please find a list of our distributors under: http://www.systec-electronic.com/distributors
Ordering Information:	+49 (0) 37 65 / 38 600-0 info@systec-electronic.com	
Technical Support:	+49 (0) 37 65 / 38 600-0 support@systec-electronic.com	
Fax:	+49 (0) 37 65 / 38 600-4100	
Web Site:	http://www.systec-electronic.com	

7th Edition Mai 2011

Table of Contents

List of Tables

List of Illustrations

1 Introduction

This manual describes in its 1st part the integration of CANopen services in PLC programs according to standard IEC61131-3. This enables the application of network variables and CANopen access through specific function blocks. Prerequisite for this is a PLC with a CANopen interface.

The functionality defined in the CiA Draft Standard 405 by the CiA (CAN in Automation e.V.) provides the basis for offering CANopen services for PLC programs according to EC1131-3. SYSTEC enhances this functionality with additional, manufacturer-specific function blocks (e.g. sending and receiving of PDOs or CAN Layer 2 messages, generation of SYNC objects).

The second part of this manual describes the use of function blocks for processing protocol-independent CAN-messages (CAN layer 2 messages) in a PLC program. By means of CAN layer 2 function blocks, the PLC can also communicate with CAN-devices, which are not CANopen compatible.

Using CANopen network variables (→ Section 3)

Network variables are the simplest form of data exchange with other CANopen nodes. Access to network variables in a PLC program means the same as accessing internal, local variables in the PLC. For the PLC programmer it is therefore unimportant, whether e.g. an input variable is assigned to a local input of the control or whether it represents an input of a decentral extensionmodule. The application of network variables only requires basic CANopen knowledge. **In general, a CANopen configurator as well as the availability of EDS files for the individual CANopen devices are usually prerequisites for integrating network variables (see section 3.1).**

With the aid of network variables it is possible:

- To extend PLC inputs and outputs by using CANopen devices
- To exchange process data between various controls in order to realize decentral automation projects
- To include, in addition to SYSTEC modules, any other specialized CANopen devices in the project planning in order to design controls for special tasks with the aid of modular standard modules.

Using CANopen function blocks (→ Section 4)

CANopen function blocks enable direct access to specific CANopen services and, therefore, provide high-level application flexibility. A CANopen configurator or EDS files are not required for their utilization. **However, the utilization of CANopen function blocks requires detailed knowledge of CANopen and its various services.**

With the aid of CANopen function blocks it is possible:

- To directly exchange data via SDO (Service Data Object) or PDO (Process Data Object) with other CANopen nodes
- To request or influence the state of other CANopen nodes
- To receive error messages of other CANopen nodes
- To enable the generation of SYNC messages

Using CAN Layer 2 function blocks (→ Section 6)

CAN Layer 2 function blocks allow for a protocol-independent exchange of data between PLC and diverse CAN-devices. Here, the PLC program directly processes the CAN-messages, which are transferred to the CAN-bus.

CAN Layer 2 function blocks allow for:

- a configuration of the CAN-interface of the control with the parameters preset by the PLC program
- Writing and processing CAN-telegrams directly through the PLC program
- Sending and receiving CAN-messages in an Extended-Frame-Format (29 bit CAN Identifier, according to CAN 2.0 B)

Note: CAN Layer 2 function blocks cannot be used simultaneously with CANopen services on the same CAN-Interface. CAN Layer 2 function blocks can only be used if the CANopen functionality for the relevant CAN-Interfaces has been disabled in advance.

Part 1

CANopen

(High Level Protocol)

2 Basics of CANopen Integration of a PLC

2.1 Differences between a PLC with and without CANopen Master

The CANopen extension of a PLC and particularly the CANopen function blocks for IEC61131 are based on various CANopen services; some of which can be active on several nodes simultaneously (e.g. PDO and SDO transfer) while others can only be exclusively executed by one node (e.g. NMT Master services). This separation is also underlined by the terms "PLC with CANopen Master" and "PLC without CANopen Master". On a PLC without CANopen Master only the non-exclusive services are available while on a PLC with CANopen Master, the exclusive services can be used additionally. See Table 1 for an overview. Depending on the available services, a varying amount of CANopen function blocks exists for the PLC program. See Table 7 in section 4.1.2 for a detailed list.

Table 1: Availability of services on the PLC with and without CANopen Master

CANopen Service	PLC without CANopen Master	PLC with CANopen Master
PDO	X	X
SDO	X	X
Heartbeat	Producer / Consumer	Producer / Consumer
Lifeguarding	Slave	Master
NMT Master	-	X
SYNC Producer	-	X

Legend:
X = Functionality available
- = Functionality not available

With SYSTEC controls, the differentiation between "PLC with CANopen Master" and "PLC without CANopen Master" occurs model-dependently either through a configuration user interface, operating elements (e.g. DIP switch) or the node address. Since each node within a CANopen network has to be assigned with a unique address, it is also ensured, while linking to the node address, that the master functionality can only be executed by one node network-wide.

With control types where activation of the master functionality occurs by selecting the node address, the master functionality of the PLC is permanently linked to the node address 20H. With these control types, only the functionality "PLC without CANopen Master" can be used on all the other node addresses.

If the network only contains one PLC, it has to be operated in master mode (e.g. node address 20H). This ensures that the PLC program can monitor the state of other nodes via the CANopen function block *CAN_GET_STATE* (see section 2.2 for basics as well as section 4.5.1 for the description of the function block). Moreover, only a PLC in master mode can generate SYNC objects (see section 4.5.8). In networks with several controls, just one PLC has to be active in the master mode.

2.2 Node Configuration via PLC with CANopen Master

A PLC with CANopen Master can configure other devices in the network. The DCF files of the IO modules which were also transferred to the control while downloading the PLC program are required for this. A PLC with CANopen Master also automatically monitors each CANopen module in the network via Heartbeat or Lifeguarding. This is also the basis for the state request using function block *CAN_GET_STATE* (see section 4.5.1). Monitoring of the node is primarily determined by the respective entries in the DCF file of the corresponding node. The alternative default configuration of Heartbeat or Lifeguarding, which is used when there is no DCF file available for the respective node, is described in section 5.5. Internally, monitoring of the node occurs via the master PLC according to the following principle:

When switching on the operating voltage (Reset) and after completion of the program download, the master PLC checks whether the object 1F81H has been created in its own object dictionary (i.e. whether the object 1F81H is contained in the DCF file of the master PLC). If this is the case, the nodes specified therein are searched for via a node scan. If object 1F81H does not exist, the control determines all the available CANopen devices in the network via a network scan, which covers the entire range for all node addresses from 1 to 127 as standard (if necessary, the node range can be limited via index 3001H in the object dictionary of the master PLC, see section 5.2). Here, it is attempted to read the index 1000H in the object dictionary of the node on each node address (device type). If the device responds, it is checked whether a DCF file on the PLC is available for this node. In this case, the device's PLC carries out configuration with the parameters specified in the DCF file. If no DCF file is available, the PLC checks whether the respective device can be monitored via Heartbeat. If the node does not support Heartbeat, availability of Lifeguarding is tested alternatively. Devices which support neither Heartbeat nor Lifeguarding cannot be monitored. A state request by the PLC program is thus not possible for the respective nodes (see below). See Figure 1 for details of the node configuration procedure via CANopen Master. And see section 5.5 for configuration details of Heartbeat and Lifeguarding.

If the control receives bootup messages from other CANopen devices (e.g. because these devices are fed via a voltage supply which is subsequently switched on) after the network scan, these nodes are configured the same way as described above.

Note: When using DCF files for configuring nodes, the user is responsible for activating node guarding (Heartbeat or Lifeguarding) accordingly. Otherwise, node monitoring and a state request by the PLC program are not possible for the respective node (see below).

If errors occur during the configuration phase, the PLC sets a respective error state and checks it when restarting the PLC program (see below). Possible causes for configuration errors are:

- A node marked as "Mandatory Slave" in the object 1F81H is not available (the absence of a node marked as "Mandatory Slave" is **not** an error) or
- An error occurred while configuring a node with the parameters specified in the DCF file (SDO Abort due to access to a non-available object, write access to a read-only object, other logical configuration errors)

During PLC program start-up the PLC at first checks whether any errors occurred during the previous configuration phase. If the configuration was successful and nothing contrary is set in object 1F80H, the Master PLC sends the NMT command "Enter Pre-Operational State", followed by "Start Remote Node" for each node (node address = 0). This prompts the CANopen nodes to send their PDOs (**P**rocess **D**ata **O**bject) once. Subsequently, the PLC waits until the PDOs have been processed and the received values have been stored in the network image before starting the PLC program. This procedure ensures the initial initialization of the network variables (see section 2.3)

Note: The waiting periods can be configured via Index 3003H in the Object Dictionary of the Master PLC, see section 5.4.

If Bit 3 ("Do not send NMT-Start Remote Node") has been set in object 1F80H or if errors occurred during the configuration phase, the PLC does not automatically switch the network into the operational state. In the Pre-operational state the PLC program can at first carry out further configurations via the SDO function blocks described in section 4.4 and finally set the network (compulsory) into the Operational state through the NMT function block described in section 4.5.2.

When exiting the PLC program, the Master PLC sends the NMT command "Enter Pre-Operational State" to show each CANopen node that PDO processing has been deactivated.

With Heartbeat as well as with Lifeguarding the node sends its current state (monitoring period) at regular intervals to the PLC with CANopen Master. This state is internally evaluated in the network layer and transferred to the PLC program when calling function block *CAN_GET_STATE*. To also enable the application of block *CAN_GET_STATE* on controls without Master, each recognized node state change is forwarded to all the other controls through a Broadcast message (Default-COBID 50H, configuration via Index 3002H, see section 5.3). A state request on controls without Master is therefore no longer possible when deactivating the Master PLC (e.g. disabling the PLC program). The function block *CAN_GET_STATE* sends the state *UNKNOWN*. This state is also returned when the respective node supports neither Heartbeat nor Lifeguarding. In this case, a state request for the respective node is not possible.

The Master PLC monitors each node in the network in order to enable the PLC program (and thus the user) to carry out a state request of the nodes. However, the Master PLC does not react in case of an error; this is the user's responsibility.

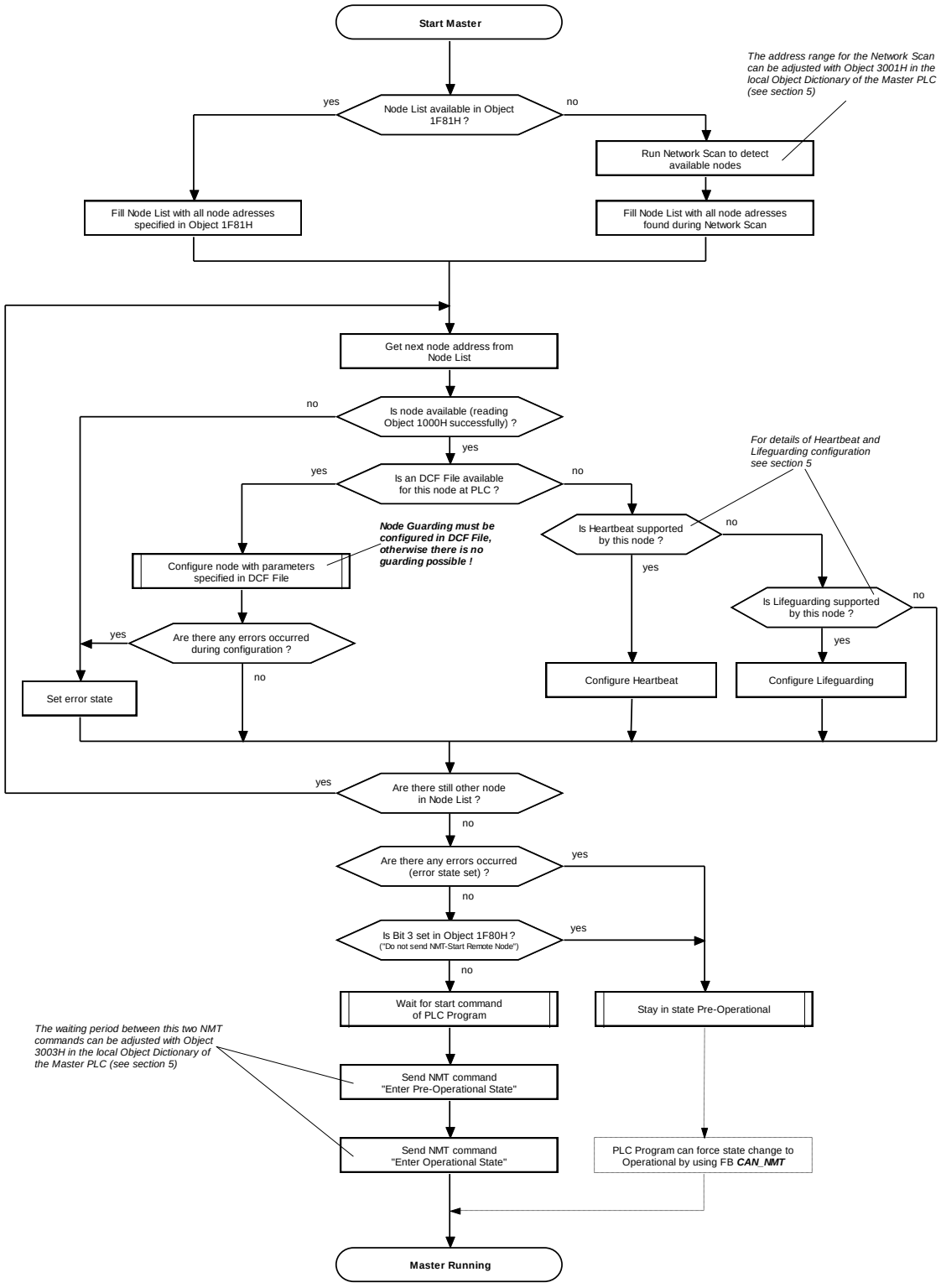


Figure 1: Node configuration procedure via CANopen Master

2.3 Initial Initialization of the Network Variables

The primary initial initialization of the network variables occurs with the initial values (entry "ParameterValue=" or "DefaultValue=") set in the DCF file of the PLC. These are used as start values for the network variables when creating the dynamic Object Dictionaries (see section 3.1).

The content of the network variables is exchanged between the individual nodes by sending PDOs (**P**rocess **D**ata **O**bject). Usually the transfer of PDOs occurs event-controlled so that the communication partners are only informed of variable changes (asynchronous transfer). This procedure reduces the bus load to the necessary minimum. However, from a CANopen device's point of view this means that it does not know the real value of a variable without further measures from the start time to the first change. Therefore, it is necessary that the Master control prompts all the CANopen devices to send their PDOs once when starting the PLC program. This way all CANopen nodes including the Master PLC know the real initial values of the input network variables. Subsequently, the notification of changes is sufficient.

The initial initialization of network variables has been realized differently on controls with and without master functionality:

PLC with CANopen Master

The only possibility supported by all CANopen devices to prompt the nodes to mandatorily send their PDOs is to change the state from Pre-Operational to Operational. To achieve this, the Master PLC at first sends the NMT command "Enter Pre-Operational State", followed by "Start Remote Node" for all nodes (node address = 0) when starting its own PLC program. This way all CANopen nodes which were active at this time in the network are prompted to send their PDOs once. **Thus, after starting a PLC program on the Master PLC all the active nodes of the network are in the Operational state.**

Note: The waiting period between the NMT command "Enter Pre-Operational State" and "Start Remote Node" can be configured via Index 3003H / Subindex 1 in the Object Dictionary of the Master PLC (see section 5.4).

The PLC independently delays the first sending of its own output network variable until the first PLC cycle has finished. The PLC program can therefore pre-assign the output network variables within the first PLC cycle with specific values which may differ from the initial values set in the DCF file (also see section 7). At the same time, it can prevent the PLC from sending initial PDOs with invalid data (initial PDOs with zero bytes).

The Master PLC sends the NMT command "Enter Operational State" with the specific node address of the respective device to CANopen IO devices which login at the network once the PLC program of the Master PLC has been started. The device is thus also prompted to transfer its PDOs with initial values. At the same time, the Master PLC also sends all its PDOs once, irrespective of whether the values therein have changed or not. This also ensures the correct initial initialization of network variables for devices which are connected with a delay.

PLC without CANopen Master

From the network's point of view a PLC without CANopen Master acts the same way as any other CANopen IO device. During start-up (switching on the operating voltage, reset) the PLC sends a bootup message and thus logs into the network. Subsequently, the PLC remains in the Pre-Operational state until it receives the NMT command "Enter Operational State" from the CANopen Master. The PLC reacts the same as any other CANopen IO device by sending its initial PDOs.

Depending on the application it might be necessary for the PLC to link the execution of its own PLC program to the respective network state (Pre-Operational / Operational). The state of the own node can be determined via the CANopen function block *CAN_GET_STATE* (see section 4.5.1). The following example illustrates the request and evaluation of the own node state:

```
VAR
    FB_CanGetState : CAN_GET_STATE;
END_VAR

CAL   FB_CanGetState (
      DEVICE := 0,                    (* own node *)
      ENABLE := TRUE)
LD    FB_CanGetState.STATE
EQ    16#0005                         (* Operational ? *)
JMPCN PreOperationalMode

OperationalMode:
(* ... *)
RET

PreOperationalMode:
(* ... *)
RET
```

3 IEC61131 Network Variables for CANopen

3.1 Basic Information for Network Variables

Numerous control units can exchange data via a CANopen network or can be extended by additional inputs and outputs via CANopen IO modules. At the PLC program layer the data exchange occurs through network variables which are declared as "VAR_EXTERNAL" according to standard IEC61131-3 and are thus marked as "outside of the control". The PLC administers a copy of these variables locally whereby the network layer is responsible for synchronizing this copy with the real value of the CANopen device. The initial initialization of the network variable described in section 2.3 is also of particular importance.

CANopen describe the PLC is a "common" IO module whose inputs and outputs are not led out at terminals but are mapped into the process image as network variables. Depending on the number and scope of network variables used in the PLC program, the appearance of the PLC changes in regard to network-sided inputs and outputs so that the same PLC can be represented differently to the CANopen network when executing different programs. To achieve this, the PLC uses a dynamic Object Dictionary (database-similar structure for the administration of variables as well as communication and mapping parameters), CANopen IO modules on the other hand usually have a static Object Dictionary.

According to the specifications in the CiA Draft Standard 405 the network variables of a PLC are created in the Object Dictionary in the range of Index A000h - AFFFh.

The following terms play a central role for further explanations of linking controls to decentral extension units:

CANopen IO module: The CANopen IO module is a unit which provides the network with certain resources, such as inputs and outputs. This type of module is a slave unit for the network management (NMT).

Mapping: The assignment variables as well as inputs and outputs to bytes or bit positions within a CAN message is called mapping.

CANopen configurator: The CANopen configurator is a special software tool which enables the planning and management of CANopen networks, the logical connection of inputs and outputs of different units, as well as the setting of network parameters. Moreover, the CANopen configurator is used to link network variables in PLC programs to the inputs and outputs of the respective CANopen IO module. **A CANopen configurator is always an external software tool which is not contained in the scope of delivery of the IEC61131 programming system *OpenPCS*.** We recommend, among others, the program "ProCANopen" from Vector Informatik.

EDS file: The EDS file (Electronic Data Sheet) is supplied by the device's manufacturer and describes the various properties of the unit, e.g. usable IOs, factory default settings for mapping and network communication, as well as the parameters which can be modified by the user.

DCF file: The DCF file (Device Configuration File) is generated by the CANopen configurator as the result of the configuration process. The configurator uses the EDS file as a "template" and amplifies it with the parameters set by the user, e.g. identifier and mapping.

The network variables used in the PLC program are linked to the inputs and outputs of CANopen IO units in order to assign decentral IOs to a PLC. Usually a CANopen configurator is required for this task. In contrast to standard IO modules there is no EDS file which specifies which inputs and outputs are available for a PLC. The end user specifies the number and types of accessible inputs and outputs

via a precise PLC program by declaring the respective network variables. Therefore, there is only one general EDS file for a PLC, which simply states that the control supports dynamic objects.

Prerequisite for the application of network variables is usually the availability of EDS files for the respective CANopen IO modules and a CANopen configurator. For simple network topologies (only a few decentral standard IO modules) it is also possible to use the predefined DCF files supplied with the programming environment *OpenPCS* (see section 3.2.3). If these prerequisites are not met, network variables cannot be used. Network communication is then only possible through the CANopen function blocks described in section 4.

3.2 Configuration Process

3.2.1 Network Configuration

The EDS file is very important for the configuration process of IO modules. The EDS file is read by the CANopen configurator to ensure that the user can access the resources provided by the CANopen IO module. In the configuration the user specifies, amongst other things, which inputs or outputs are to be used, in which bit or byte of a CAN message a respective value is transferred to the bus (mapping), and which identifier is to be used for this. As a result of this configuration process, the CANopen configurator generates a DCF file for the respective node (see Figure 2). A manual configuration of CANopen modules is only necessary if the user wishes to or has to change the standard parameters (identifier, mapping) specified by the manufacturer. The standard parameters usually result from a defined algorithm from the settable node number (Node ID) of a device and are displayed in the respective manual.

In contrast to IO modules with static inputs and outputs, a PLC uses dynamic objects, i.e. the network variables defined in the respective PLC program. But since the manufacturer of a device is unable to know the objects which are created dynamically with the operating time, there are also no specifications on this in the EDS file. Therefore, a configurator is always required for linking dynamic objects. The result of the configuration process is then stored in the DCF file. The IEC61131 programming system uses the DCF file for the PLC generated by the configurator for resolving the network variables declared as VAR_EXTERNAL and generates the required control information for the network layer from this. **The DCF file of the control is thus the central link between CANopen and the IEC61131 PLC program.** See Figure 2 for an overview of the CANopen node configuration.

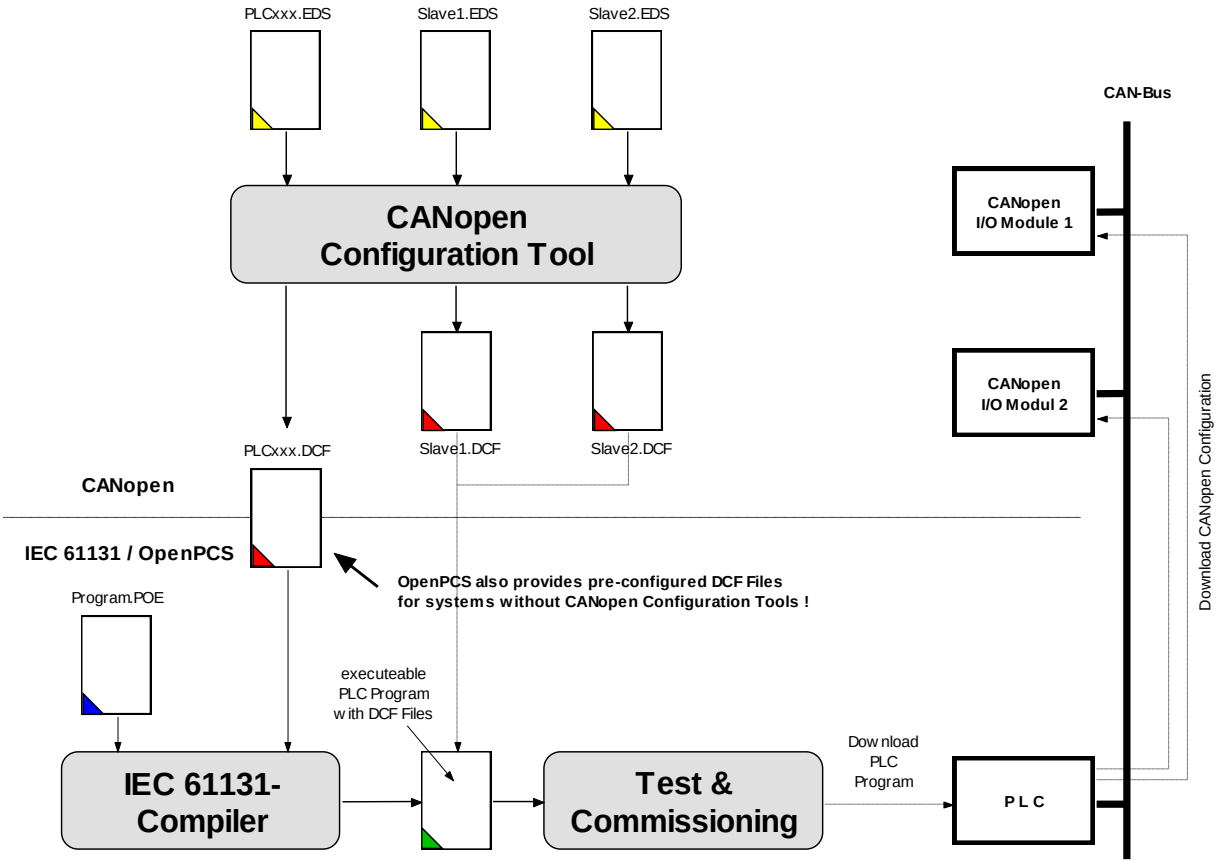


Figure 2: CANopen node configuration

The DCF files generated as result of the network configuration are assigned to the control resource in the IEC61131 programming system. The DCF file for the PLC is used for resolving the network variable declared as VAR_EXTERNAL as well as for configuring the object dictionaries of the PLC. The DCF files of the remaining IO modules are also transferred to the control and stored there during PLC program download. The CANopen Master contained in the PLC uses these DCF files to configure the IO modules accordingly in the field layer during system start. However, prerequisite for this is the activation of the master functionality of the PLC module (e.g. through DIP switch or configuration file, see the respective control manual for details).

Alternatively, it is also possible to configure the CANopen IO modules via the PLC program. It is possible to write the required parameters into the Object Dictionary of the IO modules via the SDO function blocks described in section 4.4.

Storing the EDS files for SYSTEC devices:

The EDS files for various SYSTEC devices are stored in directory *EDS-DCF* within the *OpenPCS* path (e.g. *C:\OpenPCS\EDS-DCF*). The EDS files for CANopen IO modules are also contained in the scope of delivery for these devices. Several EDS files can be selected for the PLC:

Table 2: Overview of EDS files for SYSTEC devices:

File name	Application
PLC02PDO.EDS	Unspecific/General EDS file for a small number of nodes and network variables Enables max. 2 send PDOs and 2 receive PDOs

PLC64PDO.EDS	Unspecific/General EDS file for a large number of nodes and network variables Enables max. 64 send PDOs and 64 receive PDOs
{DeviceName}_CAN0.EDS	Device-specific EDS file for CAN instance 0 (usually primary CAN instance with dynamic Object Dictionary) e.g.: PmC14_CAN0.EDS, PLCcoreCF54_CAN0.EDS
{DeviceName}_CAN1.EDS	Device-specific EDS file for CAN instance 1 (usually secondary CAN instance with static Object Dictionary) e.g.: PmC14_CAN1.EDS, PLCcoreCF54_CAN1.EDS

The EDS files differ, amongst other things, in the number of possible PDOs for data exchange between PLC and CANopen IO module. However, for controls with a dynamic Object Dictionary the number of PDOs created is determined by the precise configuration.

3.2.2 CANopen Configurator

Network parameters for the exchange of process data, e.g. send mode (synchronous, asynchronous), the used identifiers or mapping, are specified according to the demands of the user via a **CANopen configurator**. The configurator additionally enables the creation of **links between PLC and CANopen IO modules** required for network variables. The process data (usually inputs and outputs) of the IO modules is assigned here with symbolic names in order to reference them as network variables in the PLC program at a later date.

The interface between the CANopen configurator and the *OpenPCS* programming environment is the DCF file of the PLC. This has to be assigned to the respective hardware as a configuration file. The control thus receives all the network information required for the exchange of process data with CANopen IO modules.

3.2.3 Predefined DCF Files

In order to also write PLC programs without a CANopen configurator for simple network topologies (a few decentral IO modules), SYSTEC supplies predefined DCF files with the programming environment *OpenPCS* which enable the integration of up to 5 decentral SYSTEC IO modules. Overall support of third party devices cannot be guaranteed, since the modules sometimes possess individual and specific features. The PLC programmer is limited to the default configuration (identifier, mapping) of the IO modules as well as the variable names for inputs and outputs specified in the DCF file when using predefined DCF files, but the costs and time required for the procurement and familiarization with such a configuration tool no longer apply.

By using predefined DCF files it is possible to extend a PLC by the inputs and outputs of up to 5 CANopen IO modules **without a CANopen configurator**.

Prerequisite for the application of predefined DCF files is that the used CANopen IO modules support the standard configuration specified by CiA (CAN in Automation e.V.) in the CiA Draft Standards 301 and 401.

The following CANopen devices can for example be used with predefined DCF files:
- SYSTEC CANopen IO-C12 (phyPS-409-Y)
- SYSTEC CANopen IO-X1
- SYSTEC CANopen IO-X2
- SYSTEC CANopen IO-X3
- SYSTEC CANopen IO-X4
- SYSTEC CANopen-Chip164 (MM-215-Y)
- SYSTEC CANopen-ChipF40 (MM-217-Y)
- CANopen IO modules of third party manufactures which use a standard configuration according to the CiA Draft Standards 301 and 401.

Storing predefined DCF files:

The predefined DCF files are stored under the name ***DxSALVE.DCF*** in directory *EDS-DCF* within the *OpenPCS* path (e.g. *C:\OpenPCS\EDS-DCF*). The number which replaces the symbolic character "x" in the file name states the number of supported CANopen IO modules:

Table 3: Overview of predefined DCF files

File Name	Supported Save Addresses
D1SLAVE.DCF	40H
D2SLAVE.DCF	40H, 41H
D3SLAVE.DCF	40H, 41H, 42H
D4SLAVE.DCF	40H, 41H, 42H, 43H
D5SLAVE.DCF	40H, 41H, 42H, 43H, 44H

It is recommended to always integrate the DCF file as configuration file for the resource with which all required IO modules are currently covered. However, if a configuration file which supports more than the required number of IO modules is used, it does not have a negative effect on the functionality of the PLC program, but more memory space is occupied on the PLC and the administration tasks within the network layer increase, which finally leads to longer program run times.

Variables of the predefined DCF files:

When declaring the network variables in the PLC program, the symbolic names for the process data objects specified in the DCF file have to be used. In the predefined DCF files, the network variables specified in **Table 4** are defined for the CANopen devices with the node addresses 40H…44H.

Table 4: Network variables of the predefined DCF files

Variable Name	Variable Types	Access Type
IN0_IN7_xxH	BYTE, USINT, SINT	read
IN8_IN15_xxH	BYTE, USINT, SINT	read
IN16_IN23_xxH	BYTE, USINT, SINT	read
OUT0_OUT7_xxH	BYTE, USINT, SINT	write
OUT8_OUT15_xxH	BYTE, USINT, SINT	write
OUT16_OUT23_xxH	BYTE, USINT, SINT	write
AIN0_xxH	WORD, UINT, INT	read
AIN1_xxH	WORD, UINT, INT	read
AIN2_xxH	WORD, UINT, INT	read
AIN3_xxH	WORD, UINT, INT	read
AOUT0_xxH	WORD, UINT, INT	write
AOUT1_xxH	WORD, UINT, INT	write

The symbolic character string "xx" in the variable name is replaced by the respective node number of the supported CANopen device. Thus, the file *D1SLAVE.DCF* contains, e.g., the definition for the variable *IN0_IN7_40H*, and the file *D3SLAVE.DCF* defines the variables *IN0_IN7_40H*, *IN0_IN7_41H*, and *IN0_IN7_42H*.

See the respective device documentation for the actual number of usable inputs and outputs of each CANopen IO module.

Special features when using predefined DCF files:

- While digital inputs and outputs are immediately functional after the integration of predefined DCF files and without any additional configuration work, analog systems usually have to be activated. See the manual of the respective CANopen device for more information. Activation can e.g. occur during program start-up via the SDO function blocks described in section 4.4.

- Maximal configurations have been created for each PDO in the predefined DCF files (e.g. AIN0 … AIN3 for RPDO1). The individual devices usually support less objects in one PDO (e.g. only AIN0 and AIN1). As a result, emergency messages with error code 16#8210 ("PDO not processed due to length error") or error code 16#8220 ("PDO length exceeded") can appear.

3.3 Integrating DCF Files into the PLC Project

There are several ways to integrate DCF files into the PLC project. Since a CANopen configurator which combines all DCF files in a joint project is usually used to create the DCF files, this network should, if possible, also be imported into the *OpenPCS* programming environment as a consistent entity. See section 3.3.1 for a description of the procedure. Alternatively, there is also the option to create network nodes in *OpenPCS* and to manually assign them to already existing DCF files. This is, e.g., necessary when working with predefined DCF files (see section 3.2.3) or when *OpenPCS* does not recognize the format of the project file generated by the CANopen configurator. See section 3.5 for the necessary steps.

When integrating DCF files, remember that the current content of the DCF file is imported as a copy into the OpenPCS programming environment. To accept subsequently made changes to the DCF file in the PLC project, the already imported network has to be deleted from the OpenPCS programming environment and the import process has to be repeated.

3.3.1 Integrating Complete Network Projects

When integrating complete network projects, all DCF files of a CANopen network are transferred into the *OpenPCS* programming environment as a consistent entity. This is the easiest and also safest way of importing DCF files into the IEC61131 programming system. To integrate a network project into *OpenPCS,* change to "Network" view in the project browser (see Figure 3).

Proceed as described below to integrate a complete network project:

1. Right mouse click on the "PowerMap" icon (see Figure 3)

2. Select "Load File -> Import ProCANopen network" in the context menu (see Figure 3)

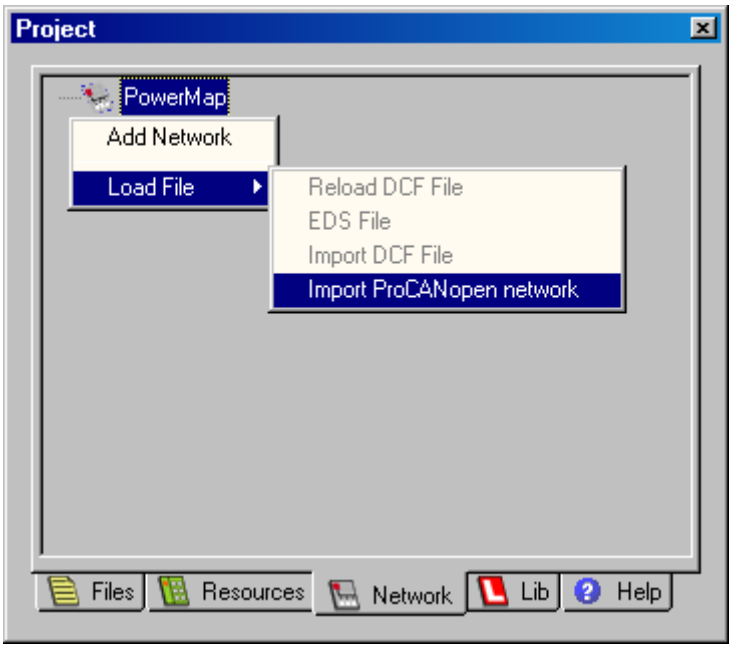


Figure 3: Importing networks into OpenPCS

3. Select the directory with the network project to be imported in the file dialog which appears and confirm it via "*OK*"

   Once the import of the network has been completed, the respective CANopen nodes are displayed in the "Network" view of the project browser (see Figure 4).

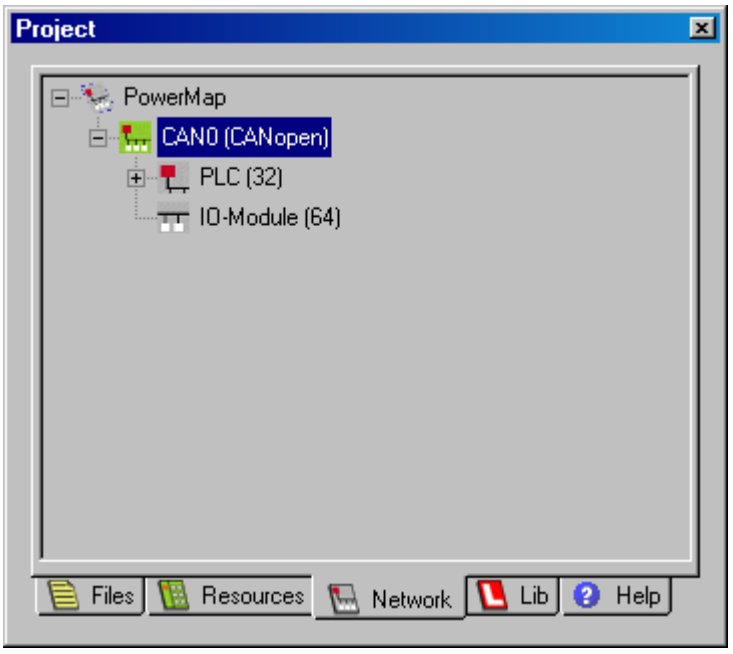


Figure 4: Presentation of the imported network in the OpenPCS project browser

4. The name of the imported network can be freely changed, if necessary. The menu item "Rename Network" in the context menu of the network node is responsible for this.

5. To link the imported network project to the active PLC resource, select the menu item "Link to Active Resource" in the context menu of the network (see Figure 5).

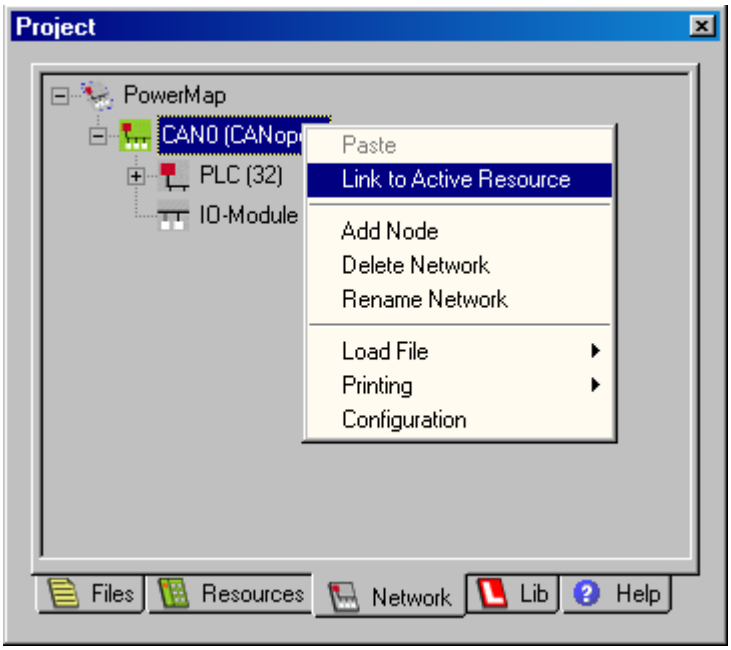


Figure 5: Linking the imported network project to the active PLC resource

See section 3.4 for the next required steps for using the network variables defined in the PLC program.

3.3.2 Manual Integration of Individual DCF Files

Manual integration of individual DCF files is an alternative to the integration of complete network projects described in section 3.3.1. Manual integration enables, e.g., the application of predefined DCF files (see section 3.2.3) as well as the import of DCF files from network projects where *OpenPCS* cannot recognize the format of the project file generated by the CANopen configurator. To integrate DCF files into *OpenPCS*, change to the "Network" view in the project browser (see section Figure 6).

Proceed as described below to manually integrate DCF files:

1. Right mouse click on the "PowerMap" icon (see Figure 6)

2. Select "Add Network" in the context menu (see Figure 6)

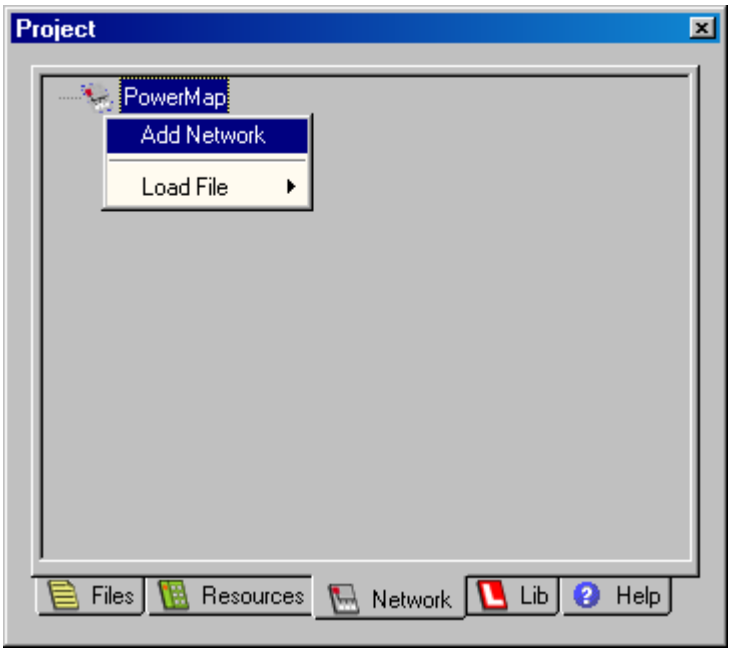


Figure 6: Creating a network in OpenPCS

3. In the dialog "Create a new file" (see Figure 7) select the category "PowerMap Network" as the "File Type" and the entry "CANopen Network" as the "Template". Enter a name for the network which is to be created in the field "Name". Finally close the dialog via "*OK*".

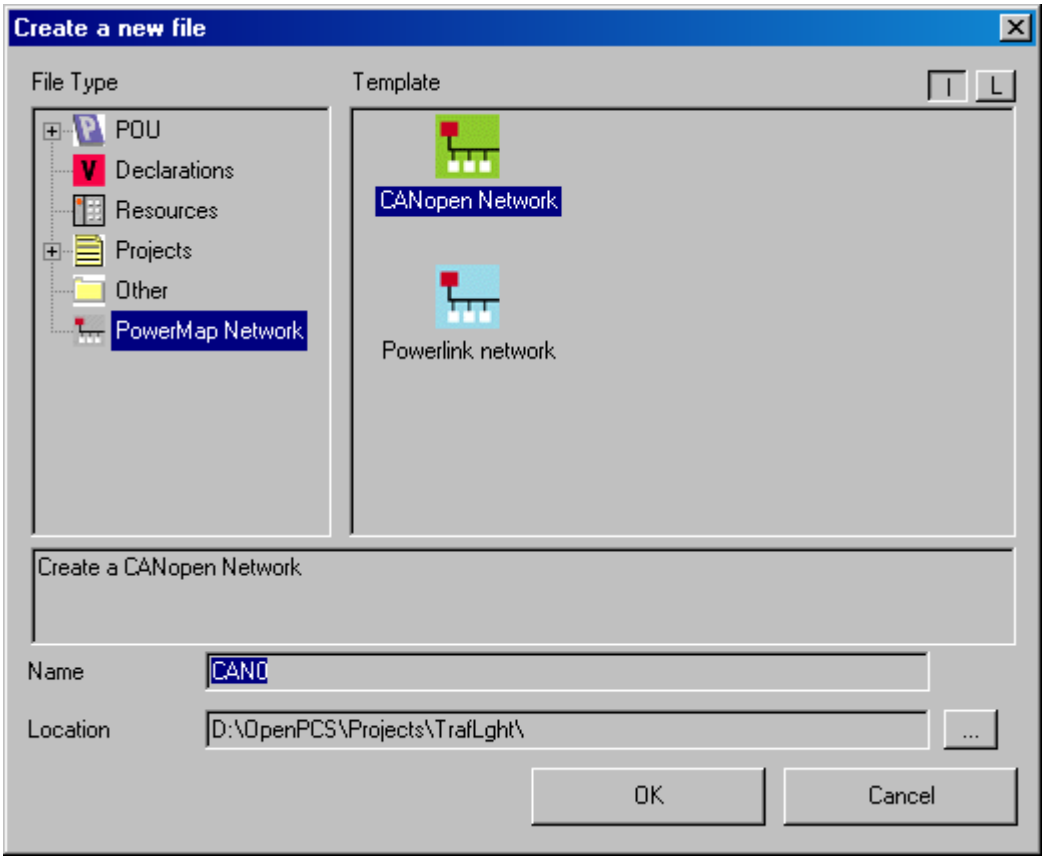


Figure 7: Creating a network entry

4. Click on the icon for the previously created network and select the menu item "Load File -> Import DCF File" in the context menu of the network (see Figure 8)

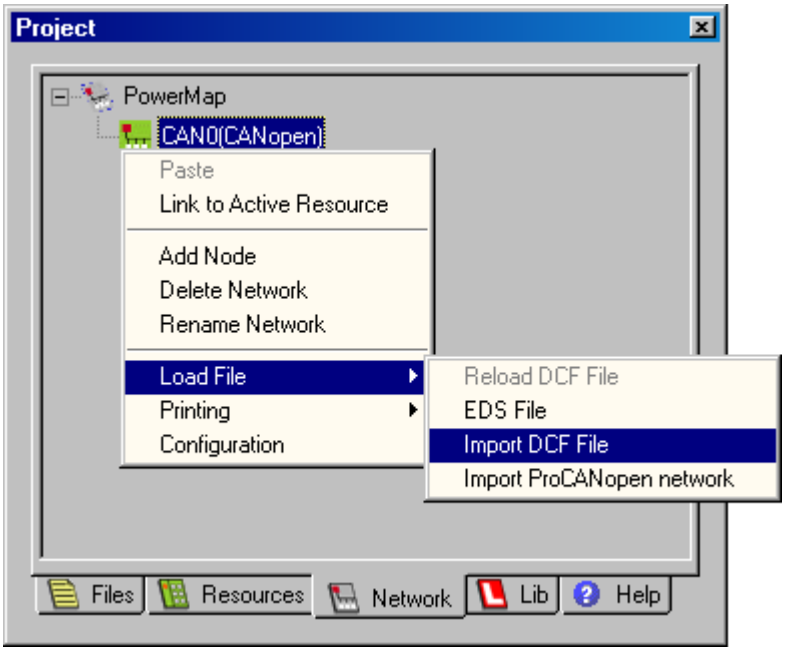


Figure 8: Adding a new network node

5. Select the DCF file to be imported in the file dialog which appears and click on the "*OK*" button to confirm the selection. This creates a new network node to which the respective DCF file is assigned. This process has to be repeated for each DCF file to be imported.

   Once the manual import of all the DCF files has been completed, the corresponding CANopen-nodes are displayed in the "Network" view of the project browser (see Figure 9).

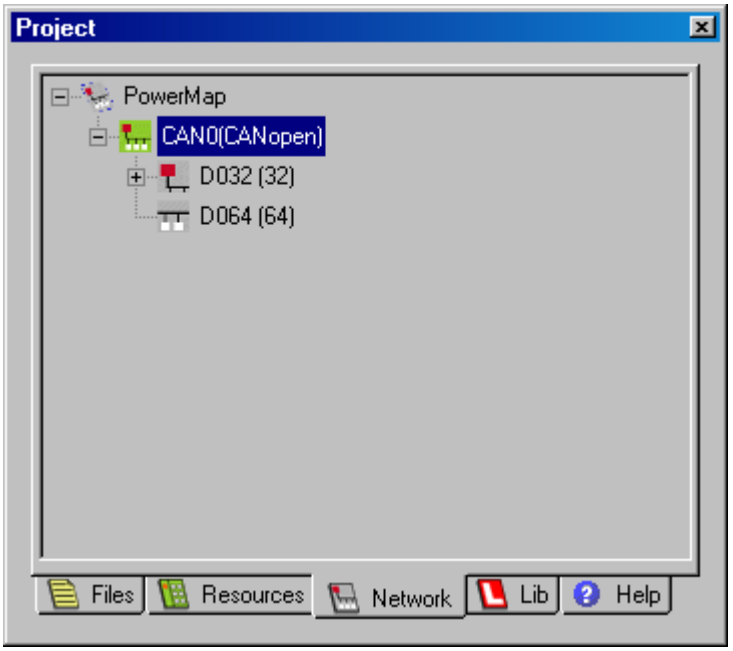


Figure 9: Presentation of the imported network nodes in the OpenPCS project browser

6. If necessary, the name of the network and the imported nodes can be changed. The menu item "Rename Network" or "Rename Node" in the context menu is used for this task (see Figure 10)

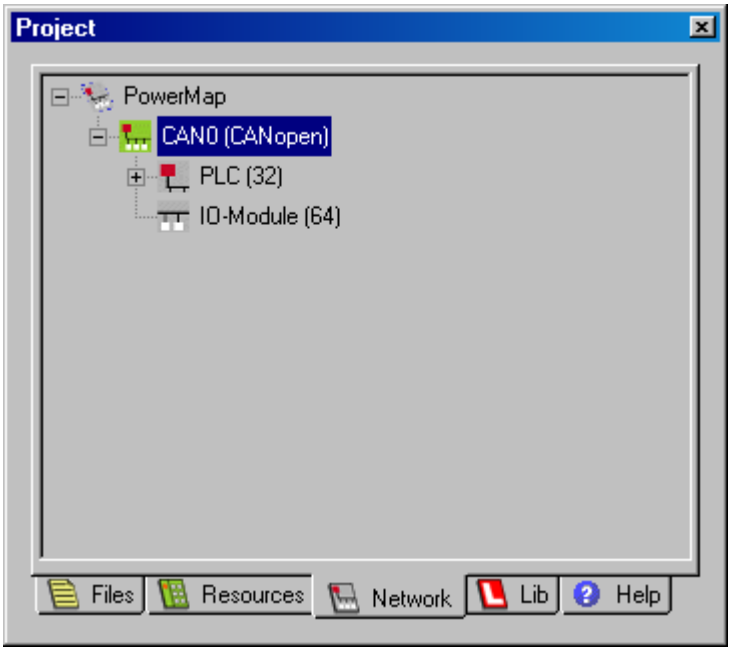


Figure 10: Presentation of the network after changing names

7. To link the network created from the manually imported DCF files to the active PLC resource, select the menu item "Link to Active Resource" in the context menu of the network (see Figure 11).

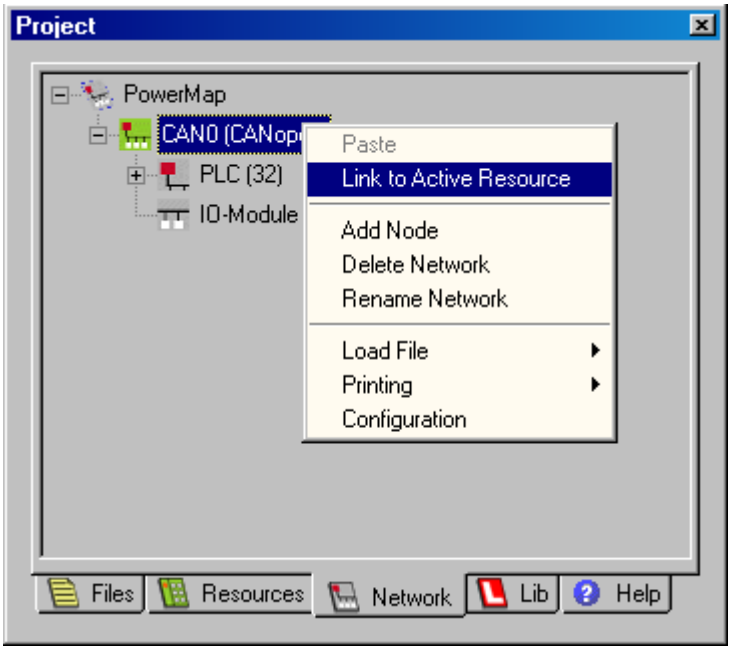


Figure 11: Linking the manually created network project to the active PLC resource

See section 3.4 for the next required steps to use the network variables defined in the network project in the PLC program.

3.4 Using Network Variables in the PLC Program

Network variables are declared in a PLC program within the key words **VAR_EXTERNAL ... END_VAR**. This means they are marked as outside of the program and thus also as "outside of the PLC". The actual declaration of the network variables does not differ from the declaration of the local variables:

```
VAR_EXTERNAL
    NetVar1 : BYTE ;
    NetVar2 : UINT ;
END_VAR
```

In addition to CANopen, there can also be other sources for external variables, e.g. EPL (Ethernet Power Link) or OPC. Therefore, the *OpenPCS* programming environment requires an assignment table in which the source of an external variable is specified.

This means that the following steps are required for the declaration of network variables in the PLC program:

1. Creating a network and integrating the required DCF files in the PLC project as described in section 3.3

2. Calling the menu item "File -> New…" in order to create a new assignment table with the aid of the dialog "Create a new file". In the dialog "Create a new file" (see Figure 12), select the category "Declarations" as the "File Type" and the entry "I/O Mapping" as the "Template". Enter a name for the assignment table which is to be created in the "Name" field. Finally close the dialog via the "*OK*" button.

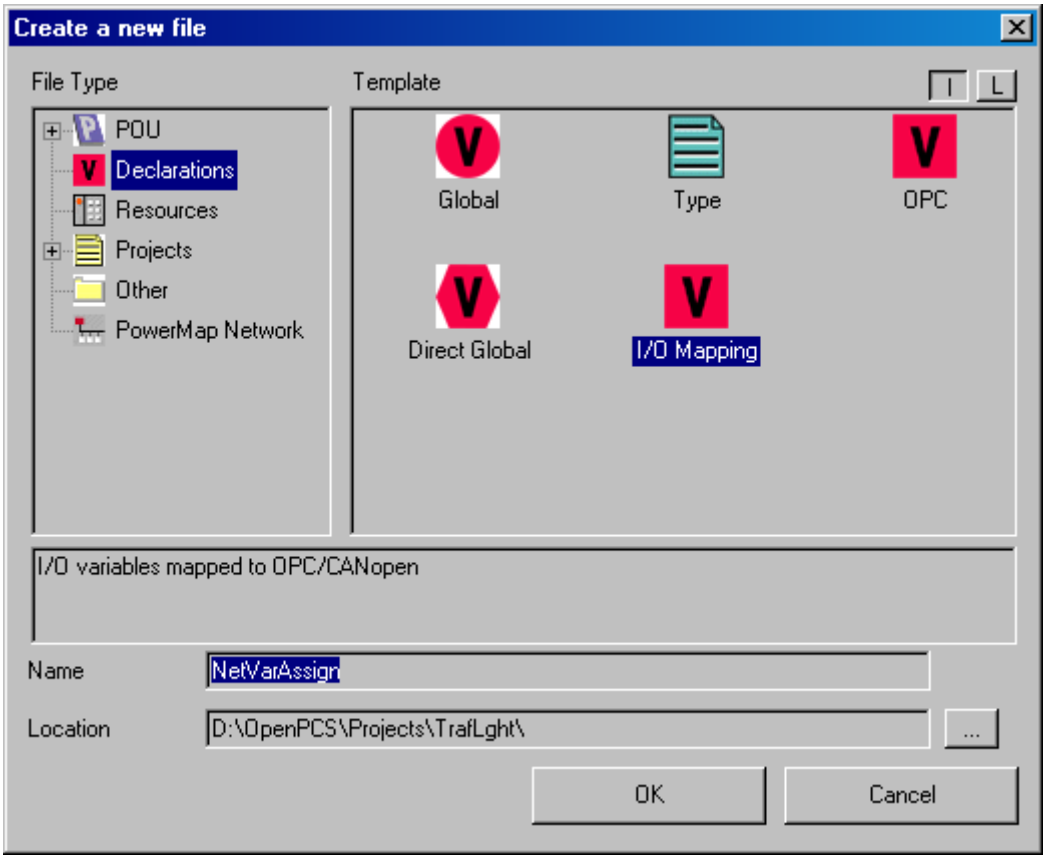


Figure 12: Creating a new assignment table

3. After creating the assignment table, the entries for the network variables have to be inserted into the assignment table from the network nodes of the PLC via double click (in this example nodes "PLC (32)") (see Figure 13). After being inserted, it is possible to adapt the name ("Name" column) and type ("IEC Type" column) of the network variables in the assignment table for utilization in the PLC program (also see the comments below).

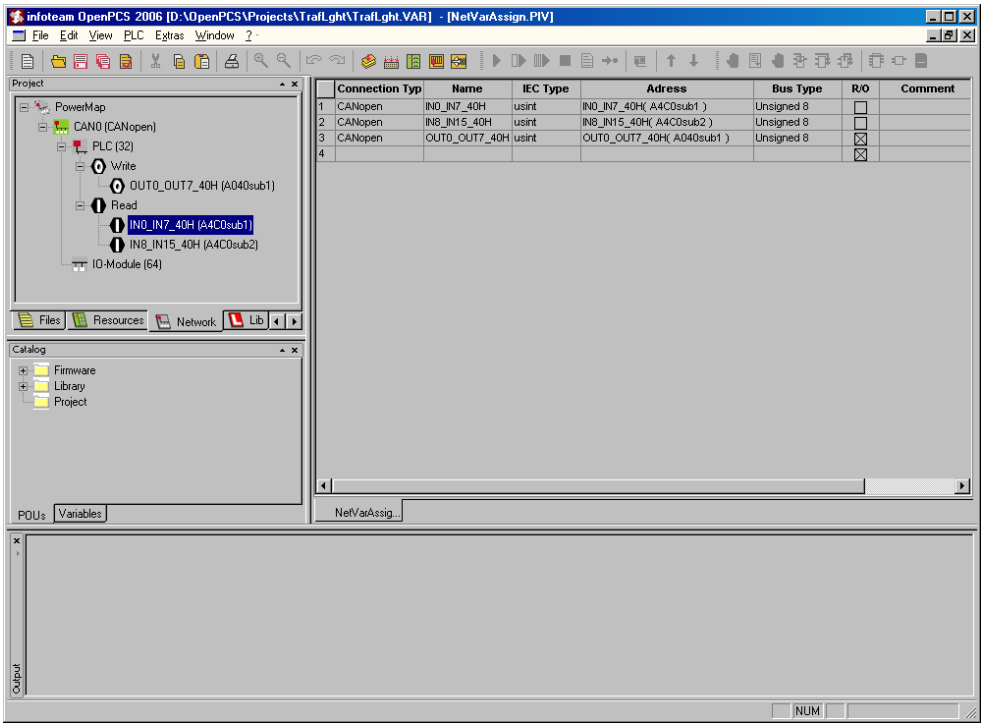


Figure 13: Inserting network variables in the assignment table

4. To link the newly created assignment table to the active PLC resource, change to the "Files" view in the project browser and then select the menu item "Link to Active Resource" in the context menu of the assignment table (see Figure 14)

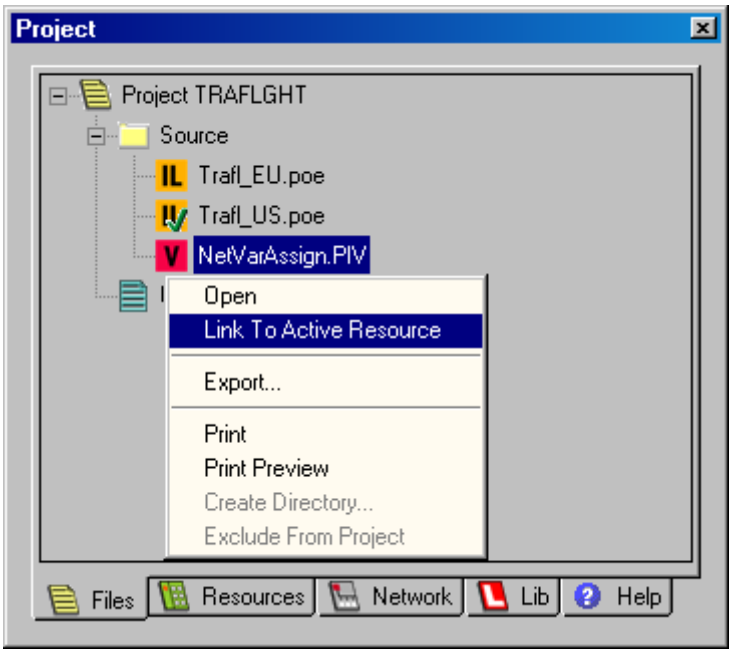


Figure 14: Linking the assignment table to the active PLC resource

5. In the "Files" view in the project browser it is possible to check whether the components required for a PLC program with network variables are linked to the active PLC resource (see Figure 15):

   - PLC program (here "TrafL_US")
   - Network ("CAN0", for creation see section 3.3)
   - Assignment table ("NetVarAssign")

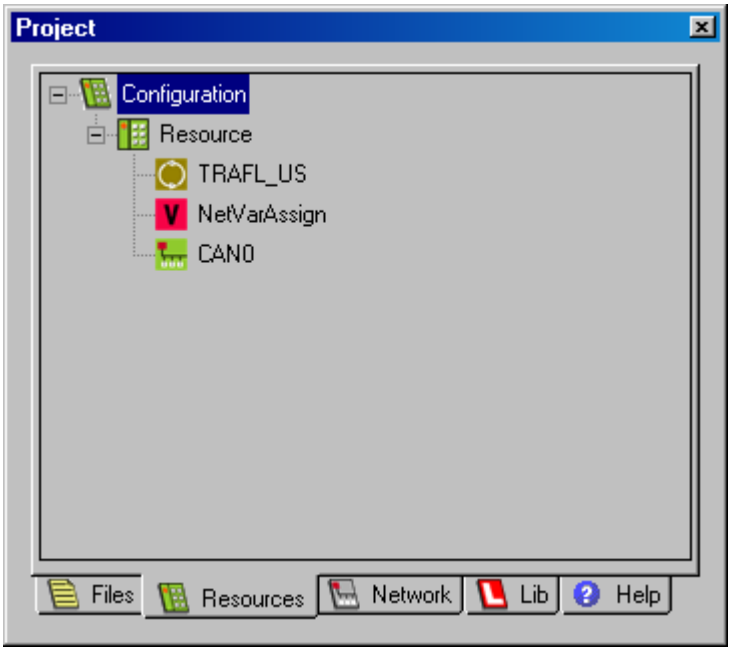


Figure 15: Presentation of the resource components in the project browser

6. Declaration of the required network variables in the PLC program within the **VAR_EXTERNAL ... END_VAR** section, the name ("Name" column) and type ("IEC Type" column) have to correspond exactly to the definition of the network variables in the assignment table, e.g.:

```
VAR_EXTERNAL
    NetVar1 : BYTE ;
    NetVar2 : UINT ;
END_VAR
```

The following points have to be considered for the declaration of the network variables:

- The **names of the network variables** have to correspond exactly between the PLC program and the assignment table. The variable name is the mutual reference point between IEC61131 and CANopen.

- A compatible data type for IEC61131 and CANopen has to be selected as the **type of network variables** (see Table 5).

- The initial values specified in the DCF file of the PLC are used as the **initial values of the network variables** (entry "ParameterValue=" or "DefaultValue="). Therefore, the renewed specification of an explicit initial value for the declaration of the network variables in the PLC program is not permitted in order to prevent discrepancies with the specifications in the DCF file. However, within the first PLC cycle the PLC program can, prior to the initial sending of specific values, assign output network variables which can deviate from the initial values specified in the DCF file (for the initial initialization of network variables also see section 2.3).

In compliance with IEC61131 a data type has to be selected according to the application (logic, arithmetic) for the declaration of network variables in the PLC program. There is no unique assignment between IEC61131 and CANopen. Table 5 contains the assignment of the data types used by IEC61131 and CANopen which can be applied as network variables.

Note that there is no unique assignment of the data types between IEC61131 and CANopen. Therefore in the PLC program the IEC61131 type has to be selected, according to the usage of this variable (logical or arithmetical operations).

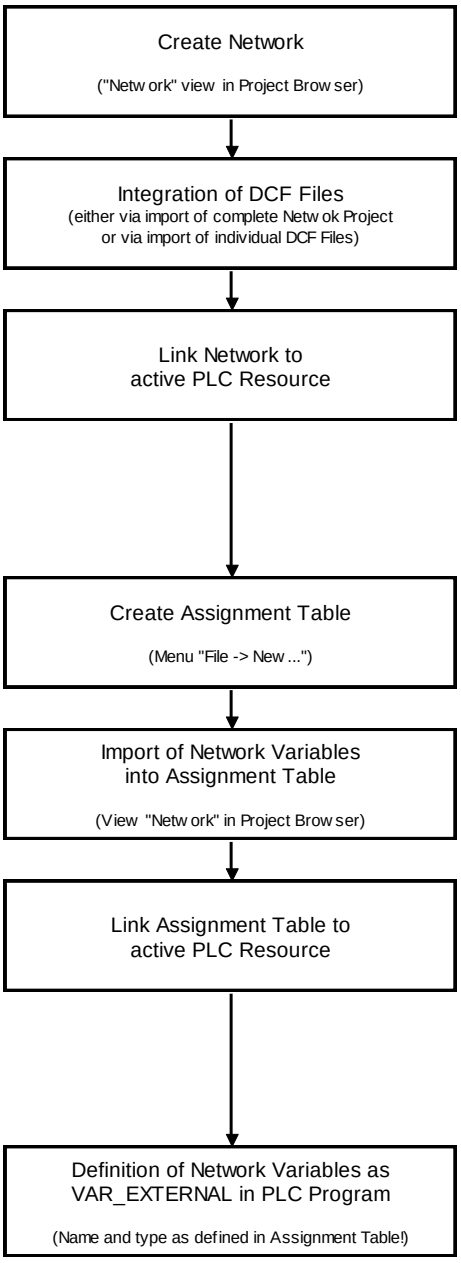
Table 5: Assignment of data types between IEC61131 and CANopen

IEC61131	CANopen	Application	Data Size (Bit)
BOOL	Boolean (*)	Logic	1
BYTE	Unsigned8	Logic	8
USINT	Unsigned8	Arithmetic (without Sign)	8
SINT	Integer8	Arithmetic (with Sign)	8
WORD	Unsigned16	Logic	16
UINT	Unsigned16	Arithmetic (without Sign)	16
INT	Integer16	Arithmetic (with Sign)	16
DWORD	Unsigned32	Logic	32
UDINT	Unsigned32	Arithmetic (without Sign)	32
DINT	Integer32	Arithmetic (without Sign)	32
REAL	Float	Arithmetic	32

The data types labeled with (*) in Table 5 are not available for all PLC types.

3.5 Summary of Required Steps

The steps described below are required to integrate network variables in a PLC program. A practical implementation is displayed in the example project in section 3.6. .



The step-by-step procedure for creating a network and for integrating the DCF files is described in section 3.3

The step-by-step procedure for creating network variables is described in section 3.4

3.6 Example Project for Network Variables

The example project *"TrafLght*l" (Traffic Light) contained in the *"SYSTEC-OpenPCS-Extension" software package* displays the integration of network variables based on the application of predefined DCF files. The file *"D1SALVE.DCF"*, which is correspondingly entered for the project configuration of the resource, is used.

The example project *"TrafLghtl"* realizes the function of a pedestrian traffic light. The PLC controls the lights for vehicles (red, amber, green) and a CANopen IO module (node address 40H) switches the signals for pedestrians (red and green). The PLC and decentral IO module are both provided with a button to start the traffic light. The traffic light cycle is started via a 0/1 intersection at one of these inputs.

For access to the inputs and outputs of the CANopen IO module, the network variables declared in *D1SLAVE.DCF* are assigned in the PLC program (*TRAFLGHT.POE*) as follows:

```
VAR_EXTERNAL
    IN0_IN7_40H : BYTE ;     (* declared in D1SLAVE.DCF *)
    OUT0_OUT7_40H : BYTE ;   (* declared in D1SLAVE.DCF *)
END_VAR
```

The logical link of the buttons at the local input of the PLC and at the input of the IO module accessible via CANopen occurs via a command sequence

```
WaitStartButton:
(* Link buttons of PLC and I/O module *)
LD  StartButton                    (* button on PLC        *)
OR  IN0_IN7_40H.0                  (* button on I/O module *)
ST  FB_StartCondition.CLK
```

The following command sequence is responsible for setting both outputs at the CANopen IO module:

```
ProgExit:
(* copy pedestrian data into network variables *)
(* for I/O module                              *)
LD  PedGreen
ST  OUT0_OUT7_40H.0
LD  PedRed
ST  OUT0_OUT7_40H.1
```

This ensures that the two local bit variables are copied into the output variable of the CANopen IO module of type *BYTE*.

4 IEC61131 Function Blocks for CANopen

4.1 Basics of CANopen Function Blocks

There are various manufacturer-specific function blocks within the IEC61131-3 to access CANopen. These contain, as a subset, the functionality defined by CiA (CAN in Automation e.V.) in CiA Draft Standard 405. Furthermore, there are additional function blocks for sending and receiving PDOs or CAN Layer 2 messages as well as for generating SYNC objects.

4.1.1 Overview of the CANopen Function Blocks

Table 6 displays an overview of the CANopen function blocks for the IEC61131-3. All the blocks have been realized as manufacturer-specific function blocks and are thus an integral part of the PLC firmware. The availability of blocks depends on the operational mode of a control. For detailed information please refer to section 4.1.2.

Table 6: Overview of the CANopen function blocks for IEC61131-3

Function Block	DS 405	Meaning	Section
CAN_GET_LOCAL_NODE_ID	yes	Request own node address	4.2.1
CAN_GET_CANOPEN_KERNEL_STATE	yes	Request CANopen kernel state of own PLC	4.2.2
CAN_REGISTER_COBID	no	Register COBID for reception of PDOs	0
CAN_PDO_READ8	no	Read received PDO	4.3.2
CAN_PDO_WRITE8	no	Send PDO	4.3.3
CAN_SDO_READ8	yes	Read object entries of a node via SDO transfer	4.4.1
CAN_SDO_WRITE8	yes	Write object entries of a node via SDO transfer	4.4.2
CAN_SDO_READ_STR	yes	Read character strings from the Object Dictionary of a node via SDO transfer	4.4.3
CAN_SDO_WRITE_STR	yes	Write character strings to the Object Dictionary of a node via SDO transfer	4.4.4
CAN_SDO_READ_BIN	yes	Read binary data from the Object Dictionary of a node via SDO transfer	4.4.5
CAN_SDO_WRITE_BIN	yes	Write binary data to the Object Dictionary of a node via SDO transfer	4.4.6
CAN_GET_STATE	yes	Request node state	4.5.1
CAN_NMT	yes	Send NMT messages	4.5.2
CAN_RECV_EMCY_DEV	yes	Read received emergency message (specific nodes)	4.5.3
CAN_RECV_EMCY	yes	Read received emergency message (any nodes)	4.5.4
CAN_WRITE_EMCY	no	Send emergency message	**4.5.5**
CAN_RECV_BOOTUP_DEV	no	Read received bootup message (specific nodes)	4.5.6
CAN_RECV_BOOTUP	no	Read received bootup message (any nodes)	4.5.7

CAN_ENABLE_CYCLIC_SYNC	no	Enable or disable cyclic SYNC messages	4.5.8
CAN_SEND_SYNC	no	Send an individual SYNC message	4.5.9

The column "CiA 405" in Table 6 indicates whether the functionality of the respective block is defined by the CiA Draft Standard 405 ("yes") or whether this block is a manufacturer-specific expansion of this standard ("no"). Please note that due to the specific peculiarities of the IEC61131 programming system implementation of blocks defined by the CiA also differs slightly from the Draft Standard 405. For general information please refer to the section 4.1.4, detailed information is contained in the respective function block section.

4.1.2 Availability of the Function Blocks on Controls with and without CANopen Master

The CANopen function blocks for IEC61131 are based on various CANopen services; some of which can be simultaneously active on several nodes (e.g. PDO and SDO transfer), while others can only be executed by a single node (e.g. NMT services). A PLC without CANopen Master only provides the non-exclusive services, while a PLC with CANopen Master also utilizes the exclusive services. The differentiation of the two PLC variants generally occurs via the node address for SYSTEC controls (see section 2.1).

Table 7 displays the availability of the individual CANopen function blocks on controls with and without Master. A PLC program can define the node address - and thus indirectly the number of usable function blocks – via function block CAN_GET_LOCAL_NODE_ID (see section 4.2.1).

Table 7: Availability of CANopen FBs on controls with and without Master

Function Block	PLC without CANopen-Master	PLC with CANopen-Master
CAN_GET_LOCAL_NODE_ID	X	X
CAN_GET_CANOPEN_KERNEL_STATE	X	X
CAN_REGISTER_COBID	X	X
CAN_PDO_READ8	X	X
CAN_PDO_WRITE8	X	X
CAN_SDO_READ8	X	X
CAN_SDO_WRITE8	X	X
CAN_SDO_READ_STR	X	X
CAN_SDO_WRITE_STR	X	X
CAN_SDO_READ_BIN	X	X
CAN_SDO_WRITE_BIN	X	X
CAN_GET_STATE	(x)	X
CAN_NMT	-	X
CAN_RECV_EMCY_DEV	-	X
CAN_RECV_EMCY	-	X
CAN_WRITE_EMCY	X	X
CAN_RECV_BOOTUP_DEV	-	X
CAN_RECV_BOOTUP	-	X
CAN_ENABLE_CYCLIC_SYNC	-	X
CAN_SEND_SYNC	-	X

Legend:
X = Functionality available
(x) = Functionality availability, but limited (please refer to the text)
- = Functionality not available

The function block *CAN_GET_STATE* can only be indirectly realized on a PLC without CANopen Master. The support of a PLC with CANopen Master is required. For detailed information please refer to sections 2.2 (description of node monitoring) and 4.5.1 (description of function block *CAN_GET_STATE).*

4.1.3 Synchronization between the CANopen Function Block and PLC Program

The majority of CANopen function blocks for IEC113-3 are executed asynchronous to the actual PLC program. The process synchronization between CANopen and PLC program occurs via the ENABLE and CONFIRM signals of the function blocks. The interaction of both signals is illustrated by Figure 16.

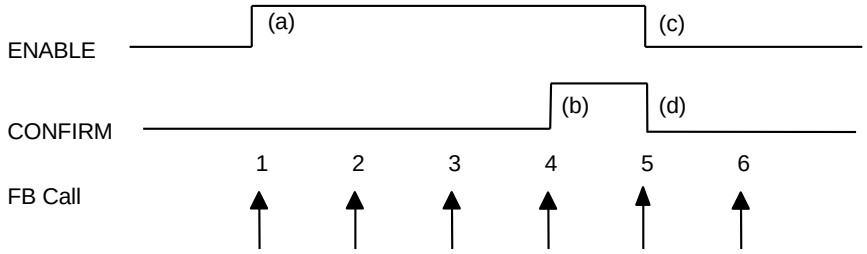


Figure 16: Process synchronization between CANopen and PLC program

The complete and successful execution of a CANopen service executed asynchronously to the PLC program occurs in the following steps:

1. Once the PLC program has installed all the input variables with valid values, it sets the input ENABLE to TRUE and calls the CANopen function block (call 1). The function block recognizes a rising edge at input ENABLE and subsequently accepts all the input values and starts the respective CANopen service (step (a)). The function block then returns to the PLC program and the initiated CANopen device is executed in the background.

2. The function block is called by the PLC program several times until realization of the CANopen service has been completed. The input ENABLE has to remain set to TRUE to enable continued execution of the CANopen service in the background (calls 2 and 3).

3. After successful completion of the CANopen service, the function block sets its output CONFIRM to TRUE. This signals to the PLC program that the service has been finished via CANopen and, if necessary, displays that further output variables are now occupied with valid values (e.g. with the data read by a node, step (b), call 4).

4. The PLC program confirms to the function block that the CANopen service has been completed by setting the input ENABLE to FALSE. Thus, the PLC program also signals that it has, if necessary, accepted the output variables supplied by the function block (step (c), call 5). During the last step, the function block sets its output CONFIRM to FALSE again, so that the system is now back in the initial state (step (d)).

The network layer usually only allows the execution of a limited number of CANopen services executed asynchronously to the PLC program at any time. The start of a service by setting the input ENABLE to TRUE (step 1) locks the respective resource for utilization by other function blocks. This locked state remains after service completion until the respective function block has been called again with input ENABLE set to FALSE (step 4) (FB sets its output CONFIRM to TRUE, step 3). The interim call of another CANopen function block results in the error message TRANSFER_BUSY at output ERROR.

Output CONFIRM only changes from FALSE to TRUE, if the respective CANopen service is completed successfully. Any occurred errors are displayed at the output ERROR or ERRORINFO. It is, therefore, necessary that a PLC program not only monitors the output CONFIRM but also the value of ERROR in order to evaluate occurred errors.

If the function block with input ENABLE set to FALSE is called, this always results in cancellation of a CANopen service active in the background and causes the internal reset of the function block. At the same time, the output CONFIRM is set to FALSE and the outputs ERROR and ERRORINFO are set to the value NO_ERROR.

4.1.4 Input/Output Parameters of the CANopen Function Blocks

The realization of the CANopen function blocks ensures they fulfill the functionality defined by the CiA Draft Standard 405. However, the used input and output parameters sometimes differ from the specification of the standard.

- Arrays or structures are not used as input or output parameters of a function block; instead the member elements contained in the array or structure are addressed as direct variables ("flat" input/output structure). This reduces the internal overhead during parameter passing.

- Instead of a specific type definition (e.g. "CIA405_DEVICE") the respective underlying elementary type (e.g. USINT) is used for the input and output parameter of the function block.

- Instead of enumerated types (e.g. "CIA405_STATE") numerical constants are used. These are listed in the section 4.1.5.

The precise input/output parameters of the CANopen function blocks that are used are described in detail in the respective function block section.

4.1.5 CANopen-Specific Constants

To identify internal error states of the network layer, the CiA Draft Standard 405 defines the specific data type "CIA405_CANOPEN_KERNEL_ERROR". This summarizes the error states which can occur within the local network layer of a PLC. These error codes are used by various function blocks as the output parameter ERROR. Table 8 lists the assignment of the used numeric constants to the corresponding error codes.

Table 8: Constants for data type "CIA405_CANOPEN_KERNEL_ERROR"

Constant		Error Code
16#0000	(= 00 dec)	NO_ERROR
16#0001	(= 01 dec)	OTHER_ERROR
16#0002	(= 02 dec)	DATA_OVERFLOW
16#0003	(= 03 dec)	TIME_OUT
16#0010	(= 16 dec)	CAN_BUS_OFF
16#0011	(= 17 dec)	CAN_ERROR_PASSIVE
16#0021	(= 33 dec)	GENERIC_ERROR (SYSTEC-specific)
16#0022	(= 34 dec)	FUNCTION_NOT_AVAILABLE
16#0023	(= 35 dec)	NO_MASTER_MODE
16#0024	(= 36 dec)	INVALID_DEVICE
16#0025	(= 37 dec)	TRANSFER_BUSY
16#0026	(= 38 dez)	INVALID_NETNUMBER
16#0030	(= 48 dec)	NO_SDO_CHANNEL_AVAILABLE
16#0031	(= 49 dec)	SDO_BUSY
16#0032	(= 50 dec)	SDO_INITIALIZE_ERROR

16#0033	(= 51 dec)	SDO_LENGTH_ERROR
16#0034	(= 52 dec)	SDO_ERROR (SDO Abort Code at ERRORINFO)
16#0040	(= 64 dec)	NO_VALID_DATA_AVAILABLE
16#0041	(= 65 dec)	COBID_ALREADY_REGISTERED
16#0042	(= 66 dec)	NO_FREE_COBID_TABLE_ENTRY
16#0043	(= 67 dec)	NO_SUCH_COBID_REGISTERED
16#0044	(= 68 dec)	NO_FREE_RECEIVE_CHANNEL
16#0045	(= 69 dec)	DATA_LENGTH_ZERO_NOT_ALLOWED

To identify the current state of the CANopen device, the CiA Draft Standard 405 defines the specific data type "CIA405_STATE". Table 9 lists the assignment of the used numeric constants to the respective state values. The state values UNKNOWN and NOT_AVAIL are an extension of the respective definitions of the CiA Draft Standard 301; all the other constant values correspond with this standard.

Table 9: Constants for data type "CIA405_STATE"

Constant	State Value
16#0000	INIT
16#0001	RESET_COMM
16#0002	RESET_APP
16#0003	PRE_OPERATIONAL
16#0004	STOPPED
16#0005	OPERATIONAL
16#0006	UNKNOWN
16#0007	NOT_AVAIL

To identify the state in which a CANopen device should change, the CiA Draft Standard 405 defines the specific data type "CIA405_TRANSITION_STATE". Table 10 lists the assignment of the used numeric constants to the respective state values. The constant values correspond with the respective definitions of the CiA Draft Standard 301.

Table 10: Constants for data type "CIA405_TRANSITION_STATE"

Constant	State Value
16#0000	START_REMOTE_NODE
16#0001	STOP_REMOTE_NODE
16#0002	ENTER_PRE_OPERATIONAL
16#0003	RESET_NODE
16#0004	RESET_COMMUNICATION

The CiA Draft Standard 405 furthermore defines the specific data types "CIA405_SDO_ERROR" as well as "EMCY_ERR_CODE" and "EMCY_ERR_REGISTER". These data types represent the error message generated by another node.

The SYSTEC specific data type "CAN_SDO_TYPE" is used with some SDO function blocks to select the SDO transfer mode. Table 11 lists the assignment of the used numeric constants to the respective SDO types.

Table 11: Constants for data type "CAN_SDO_TYPE"

Constant	SDO Type
0	SDO_TYPE_AUTO_BEST_CASE
1	SDO_TYPE_SEGMENTED_TRANSFER
2	SDO_TYPE_BLOCK_TRANSFER

The data type "CIA405_SDO_ERROR" is used for the parameter ERRORINFO of the SDO function blocks and provides the SDO Abort Code of the communication partner. The general Abort Codes are defined in the CiA Draft Standard 301, but can be extended by the manufacturer of the respective CANopen module. Table 12 lists the assignment of the used numeric constants to the typical SDO Abort Codes.

Table 12: Constants for data type "CIA405_SDO_ERROR"

Constant	SDO Abort Code
16#05030000	TOGGEL_BIT_ERROR
16#05040000	TIME_OUT
16#05040001	UNKNOWN_COMMAND_SPECIFIER
16#05040002	INVALID_BLOCK_SIZE
16#05040003	INVALID_SEQUENCE_NUMBER
16#05040004	CRC_ERROR
16#05040005	OUT_OF_MEMORY
16#06010000	UNSUPPORTED_ACCESS
16#06010001	READ_TO_WRITE_ONLY_OBJ
16#06010002	WRITE_TO_READ_ONLY_OBJ
16#06020000	OBJECT_NOT_EXIST
16#06040041	OBJECT_NOT_MAPPABLE
16#06040042	PDO_LENGTH_EXCEEDED
16#06040043	GEN_PARAM_INCOMPATIBILITY
16#06040047	GEN_INTERNAL_INCOMPATIBILITY
16#06060000	ACCESS_FAILED_DUE_HW_ERROR
16#06070010	DATA_TYPE_LENGTH_NOT_MATCH
16#06070012	DATA_TYPE_LENGTH_TOO_HIGH
16#06070013	DATA_TYPE_LENGTH_TOO_LOW
16#06090011	SUB_INDEX_NOT_EXIST
16#06090030	VALUE_RANGE_EXCEEDED
16#06090031	VALUE_RANGE_TOO_HIGH
16#06090032	VALUE_RANGE_TOO_LOW
16#06090036	MAX_VALUE_LESS_MIN_VALUE
16#08000000	GENERAL_ERROR
16#08000020	DATA_NOT_TRANSF_OR_STORED
16#08000021	DATA_NOT_TRANSF_DUE_LOCAL_CONTROL
16#08000022	DATA_NOT_TRANSF_DUE_DEVICE_STATE
16#08000023	OBJECT_DICTIONARY_NOT_EXIST

The data types "EMCY_ERR_CODE" and "EMCY_ERR_REGISTER" are used for the respective parameters of function blocks CAN_RECV_EMCY and CAN_RECV_EMCY_DEV. They contain the emergency error information of the node which generates the respective emergency message. The general emergency errors are defined in the CiA Draft Standard 301, but can be extended by the manufacturer of the respective CANopen module.

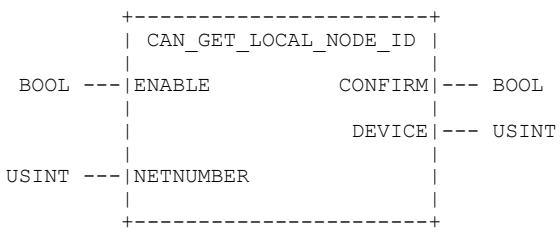
4.2 Function Blocks for Accessing the Local CANopen Kernel

The function blocks for accessing the local CANopen kernel of your own PLC enable the request of the node address as well as the state of the network layer. These function blocks do not require communication with other nodes.

4.2.1 Function Block CAN_GET_LOCAL_NODE_ID

FB for requesting the local node address.

Prototype of the Function Block



Definition of Operands

DEVICE	Local node address of the PLC
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC 61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

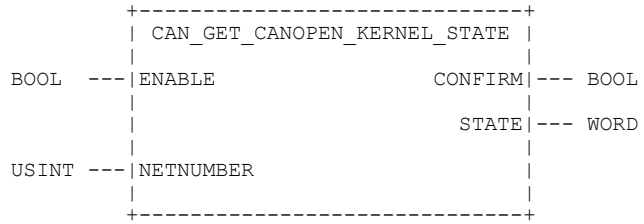
Description

The function block CAN_GET_LOCAL_NODE_ID is used to request the local node address of the PLC. The node address of a control influences the availability of the various function blocks (PLC with and without CANopen Master, see section 4.1.2)

4.2.2 Function Block CAN_GET_CANOPEN_KERNEL_STATE

FB for state request of the CANopen kernel of your own PLC.

Prototype of the Function Block



Definition of Operands

STATE	State or error code according to the data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8 in section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_GET_CANOPEN_KERNEL_STATE is used to request the state of the CANopen kernel of your own PLC.

4.3 Function Blocks for PDOs and CAN Layer 2

From the PLC's point of view PDOs and CAN Layer 2 messages are generated by a transmitter at any time. Their reception does not depend on the execution of the PLC program, but occurs asynchronously. Therefore, the reception of these messages cannot be controlled or influenced by calling a function block. Processing of PDOs and CAN Layer 2 messages via a PLC program thus requires the buffering of the asynchronously received messages in the network layer until function block CAN_PDO_READ8 has been called (see section 4.3.2). PLC program-relevant messages should be filtered here due to the number of possible CAN messages. To achieve this, the CAN identifiers (COBIDs) of the relevant messages are at first registered via function block CAN_REGISTER_COBID (see section 0). Reception, and therefore access via the function block CAN_PDO_READ8, is only possible for registered messages.

The function blocks for PDOs and CAN Layer 2 messages manufacturer-specifically extend the functional scope of available CANopen services defined by the CiA Draft Standard 405.

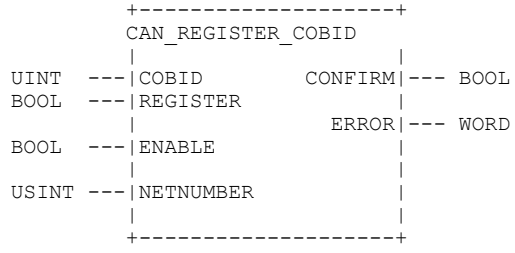
Note: The here described function blocks CAN PDO_READ8 (section 4.3.2) and CAN_PDO_WRITE8 (section 4.3.3) allow for an integration of devices in existing CANopen-networks, which are not CANopen-capable.They have been designed as a supplement to the function blocks defined in the CiA Draft Standard 405 and are therefore subject to the restrictions given by CANopen (e.g. selection of useable CAN-identifiers)

A non-restricted data exchange on CAN Layer 2 level without CANopen restrictions allows for the CAN Layer 2 function blocks described in the second part of this manual (see section 6)

4.3.1 Function Block CAN_REGISTER_COBID

FB for registering or deleting the reception of PDOs and CAN Layer 2 messages via the network layer.

Prototype of the Function Block



Definition of Operands

COBID COBID (CAN identifier) of the new message to be entered into the registry or to be deleted from the registry (0 for deleting all messages)

REGISTER	TRUE = Enter COBID into registry FALSE = Delete COBID from registry
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

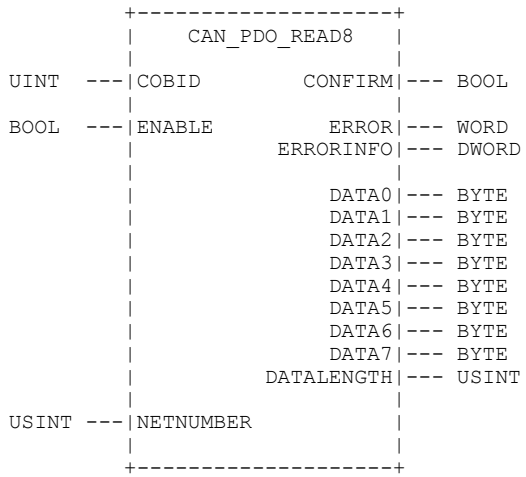
The function block CAN_REGISTER_COBID is used to register a PDO or a CAN Layer 2 message for reception by the network layer or to delete such a registration. When calling the block with input REGISTER set to TRUE, the specified COBID (CAN identifier) for receiving messages in the network layer is registered. When calling with REGISTER = FALSE, registry of the respective COBIDs is deleted again. Calling the block with REGISTER = FALSE and COBID = 0 deletes all registries and all the messages stored in the buffer of the network layer.

Basically, the network layer only supports access to PDOs and CAN Layer 2 messages by calling function block CAN_PDO_READ8 (see section 4.3.2) for registered messages.

4.3.2 Function Block CAN_PDO_READ8

FB for reading PDOs and CAN Layer 2 messages from the receive buffer of the network layer.

Prototype of the Function Block



Definition of Operands

COBID	COBID (CAN identifier) of the message to be read (PDO or CAN Layer 2)
DATA0 - DATA7	Data bytes of the received CAN message
DATALENGTH	Length of the received CAN message
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)

ERRORINFO	Reserved for additional error information
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

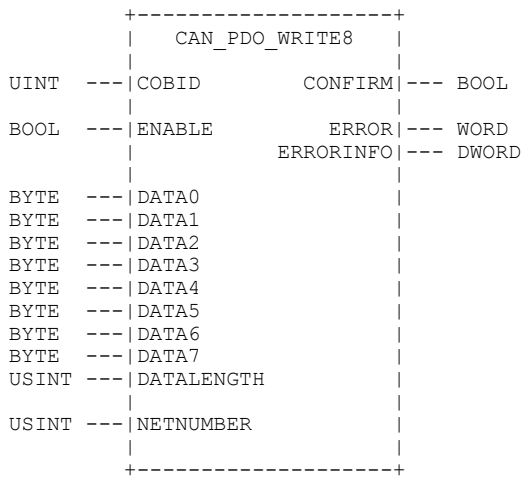
The function block CAN_PDO_READ8 is used for reading PDOs and CAN Layer 2 messages from the receive buffer of the network layer. It is only supported to access messages which have been registered via function block CAN_REGISTER_COBID (see section 0). If several messages with the same COBID are received between to subsequent calls of CAN_PDO_READ8, the last received message overwrites the previous one. This way the current message always remains in the receive buffer. After reading via the function block CAN_PDO_READ8, the respective message is deleted from the receive buffer of the network layer. This prevents repeated reading of a message by the PLC program.

If the output CONFIRM has been set to TRUE when the function block returns, the elements DATA0 to DATA7 contain the individual bytes of the received message. Output DATALENGTH states the number of valid data bytes (from DATA0). But if output CONFIRM has been set to FALSE, the receive buffer in the network layer does not contain messages with the specified COBID. With CONFIRM it is thus possible to distinguish whether a valid message with a 0 byte length or without a message has been received. An unavailable message is also displayed via the error code NO_VALID_DATA_AVAILABLE at output ERROR (see Table 8, section 4.1.5).

4.3.3 Function Block CAN_PDO_WRITE8

FB for sending PDOs and CAN Layer 2 messages via the network layer.

Prototype of the Function Block



Definition of Operands

COBID	COBID (CAN identifier) of the message to be sent (PDO or CAN Layer 2)
DATA0 - DATA7	Data bytes of the CAN message to be sent
DATALENGTH	Length of the CAN message to be sent
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
ERRORINFO	Reserved for additional error information
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_PDO_WRITE8 is used for sending PDOs and CAN Layer 2 messages via the network layer. The individual bytes of the message to be sent have to be transferred to the elements DATA0 to DATA7. Output DATALENGTH states the number of valid data bytes (from DATA0).

When calling the function block CAN_PDO_WRITE8 the message to be sent is stored in the send buffer of the CANopen kernel. If no error occurs during this phase (message could be successfully stored in the send buffer), the block whose output CONFIRM has been set to TRUE returns. However, there is no feedback to the PLC program stating whether the message has been sent successfully or not.

4.4 Function Blocks for SDOs

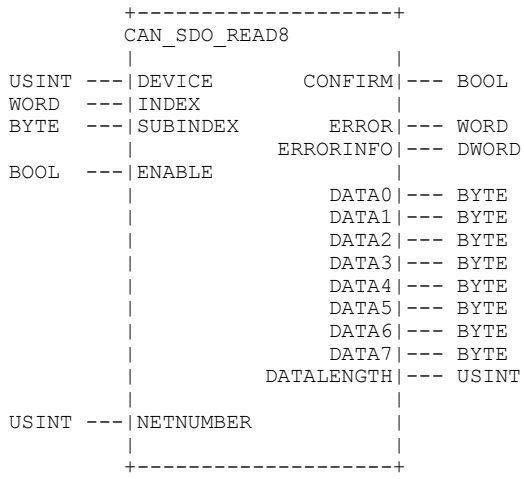
A PLC can write or read entries in the Object Dictionary of a node via SDOs. Data transfer takes place in confirmation mode to ensure that communication errors can be recognized reliably. Since the complete SDO transfer always consists of several CAN messages, it has to take place asynchronously in the background in order not to block the PLC program. The function blocks for SDOs immediately return after the transfer has been initialized. The procedure described in section 4.1.3 is used to synchronize between the CANopen function block and the PLC program using the parameters ENABLE and CONFIRM.

The network layer provides an SDO channel to be used by the PLC program. This ensures that only one SDO function block can be active at a time. Upon successful initialization of the SDO transfer, the SDO channel is locked and cannot be used by other blocks. This lock state remains until the active SDO function block is recalled with input ENABLE set to FALSE after completion of the data transfer (see section 4.1.3).

4.4.1 Function Block CAN_SDO_READ8

FB for reading object entries of a node via SDO transfer.

Prototype of the Function Block



Definition of Operands

DEVICE	Address of the node to be read (1-127 or 0 for local OD)
INDEX	Number of the index entry to be read
SUBINDEX	Number of the subindex entry to be read
DATA0 - DATA7	Data bytes of the read entry
DATALENGTH	Length of the read entry
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
ERRORINFO	SDO Abort Code of the communication partner according to data type "CIA405_SDO_ERROR" (see Table 12, section 4.1.5)

NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_SDO_READ8 is used for reading object entries of a node by using the SDO transfer. The SDO transfer always takes place in the background; the procedure described in section 4.1.3 therefore has to be used for synchronization between the function block and PLC program, using the parameters ENABLE and CONFIRM.

If the output CONFIRM has been set to TRUE when the function block returns, the elements DATA0 to DATA7 contain the individual bytes of the read object entry. Output DATALENGTH states the number of valid data bytes (from DATA0).

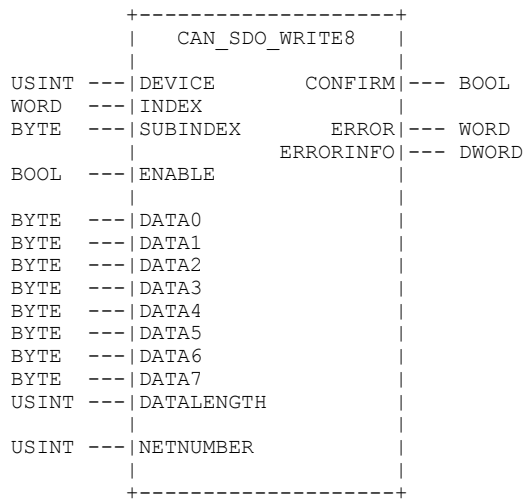
The network layer only supports a limited number of SDO transfers via the PLC program at a time (Standard: max. 5 transfers, if there is no deviating entry in the manual of the PLC). After starting the SDO transfer by setting ENABLE to TRUE, the respective SDO channel is locked and cannot be used by other blocks. The lock state remains until the SDO function block has been recalled with input ENABLE set to FALSE (see section 4.1.3). If calling through ENABLE = FALSE is skipped, the resource remains permanently locked and is no longer available for the PLC.

The local Object Dictionary of the PLC can be accessed by calling the function block with DEVICE = 0. This way it is also possible to read values from your own OD.

4.4.2 Function Block CAN_SDO_WRITE8

FB for writing object entries of a node via SDO transfer.

Prototype of the Function Block



Definition of Operands

DEVICE	Address of the node to be written (1-127 or 0 for local OD)
INDEX	Number of the index entry to be written
SUBINDEX	Number of the subindex entry to be written
DATA0 - DATA7	Data bytes of the entry to be written
DATALENGTH	Length of the entry to be written
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
ERRORINFO	SDO Abort Code of the communication partner according to the data type "CIA405_SDO_ERROR" (see Table 12, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_SDO_WRITE8 is used for writing object entries of a node, using the SDO transfer. The SDO transfer always takes place in the background; the procedure described in section 4.1.3 therefore has to be used for synchronization between the function block and PLC program, using the parameters ENABLE and CONFIRM.

The individual bytes of the object entry to be written have to be transferred to the elements DATA0 to DATA7. Input DATALENGTH states the number of valid data bytes (from DATA0).

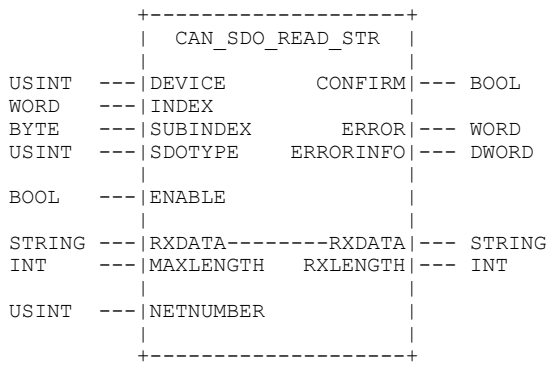
The network layer only supports a limited number of SDO transfers via the PLC program at a time (Standard: max. 5 transfers, if there is no deviating entry in the manual of the PLC). After starting the SDO transfer by setting ENABLE to TRUE, the respective SDO channel is locked and cannot be used by other blocks. The lock state remains until the SDO function block has been recalled with the input ENABLE set to FALSE (see section 4.1.3). If calling through ENABLE = FALSE is skipped, the resource remains permanently locked and is no longer available for the PLC.

The local Object Dictionary of the PLC can be accessed by calling the function block with DEVICE = 0. This way it is also possible to write values in your own OD.

4.4.3 Function Block CAN_SDO_READ_STR

FB for reading character strings from the Object Dictionary of a node via SDO transfer.

Prototype of the Function Block



Definition of Operands

DEVICE	Address of the node to be read (1-127 or 0 for local OD)
INDEX	Number of the index entry to be read
SUBINDEX	Number of the subindex entry to be read
SDOTYPE	Type of the SDO transfer mode to be used according to data type "CAN_SDO_TYPE" (see Table 11, section 4.1.5) Note: If no value is explicitly assigned to this input (input open or not used), the function block uses mode "SDO_TYPE_AUTO_BEST_CASE". The network layer individually selects the best suitable SDO transfer mode based on the amount of data to be transferred.
RXDATA	String variable for receiving the read characters
MAXLENGTH	Limitation of the number of characters to be read. If the number is 0, the buffer length of the transferred string is internally determined and used as the delimiter for the number of characters to be read (Note: the standard buffer size of a string in OpenPCS is 32 characters).
RXLENGTH	Length of the read character string
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
ERRORINFO	SDO Abort Code of the communication partner according to data type "CIA405_SDO_ERROR" (see Table 12, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_SDO_READ_STR is used for reading character strings from the Object Dictionary of a node by using the SDO transfer. The SDO transfer always takes place in the background; the procedure described in section 4.1.3 has to be, therefore, used for synchronization between the function block and PLC program, using the parameters ENABLE and CONFIRM.

If output CONFIRM has been set to TRUE when the function block returns, the string transferred as element RXDATA contains the character string of the read object entry. Output RXLENGTH states the number of read characters (equals LEN(RXDATA);).

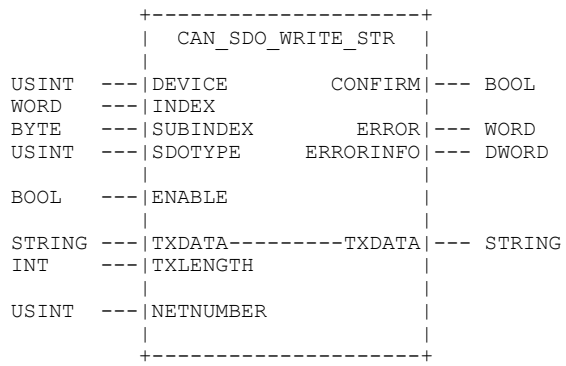
The network layer only supports a limited number of SDO transfers via the PLC program at a time (Standard: max. 5 transfers, if there is no deviating entry in the manual of the PLC). After starting the SDO transfer by setting ENABLE to TRUE, the respective SDO channel is locked and cannot be used by other blocks. The lock state remains until the SDO function block has been recalled with input ENABLE set to FALSE (see section 4.1.3). If calling through ENABLE = FALSE is skipped, the resource remains permanently locked and is no longer available for the PLC.

The local Object Dictionary of the PLC can be accessed by calling the function block with DEVICE = 0. This way it is also possible to read values from your own OD.

4.4.4 Function Block CAN_SDO_WRITE_STR

FB for writing character strings into the Object Dictionary of a node via SDO transfer.

Prototype of the Function Block



Definition of Operands

DEVICE	Address of the node to be written (1-127 or 0 for local OD)
INDEX	Number of the index entry to be written
SUBINDEX	Number of the subindex entry to be written
SDOTYPE	Type of the SDO transfer mode to be used according to data type "CAN_SDO_TYPE" (see Table 11, section 4.1.5) Note: If no value is explicitely assigned to this input (input open or not used), the function block uses mode "SDO_TYPE_AUTO_BEST_CASE". The network layer individually selects the best suitable SDO transfer mode based on the amount of data to be transferred.
TXDATA	String variable with the character string to be written
TXLENGTH	Number of characters to be written; if the number is 0, the length of the character string contained in the string TXDATA is internally determined (equals LEN(TXDATA);) and used as the number of characters to be written.
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
ERRORINFO	SDO Abort Code of the communication partner according to data type "CIA405_SDO_ERROR" (see Table 12 , section 4.1.5)

NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_SDO_WRITE_STR is used for writing character strings into the Object Dictionary of a node by using the SDO transfer. The SDO transfer always takes place in the background; the procedure described in section 4.1.3 therefore has to be used for synchronization between the function block and PLC program, using the parameters ENABLE and CONFIRM.

The character string to be written in to the Object Dictionary has to be transferred to element TXDATA. Input TXLENGTH specifies the number of valid characters. If this value is 0, the length of the character string contained in string TXDATA is internally determined (equals LEN(TXDATA);) and used as the number of characters to be written. In this case, the string content occupied entirely is written.

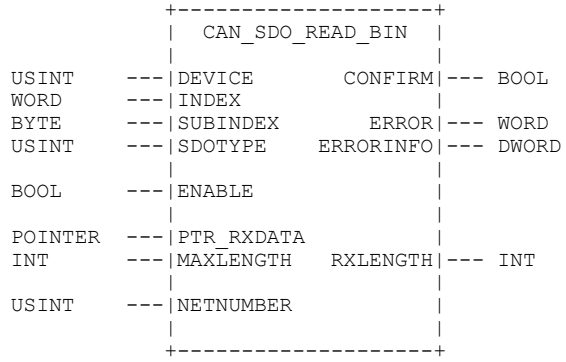
The network layer only supports a limited number of SDO transfers via the PLC program at a time (Standard: max. 5 transfers, if there is no deviating entry in the manual of the PLC). After starting the SDO transfer by setting ENABLE to TRUE, the respective SDO channel is locked and cannot be used by other blocks. The lock state remains until the SDO function block has been recalled with input ENABLE set to FALSE (see section 4.1.3). If calling through ENABLE = FALSE is skipped, the resource remains permanently locked and is no longer available for the PLC.

The local Object Dictionary of the PLC can be accessed by calling the function block with DEVICE = 0. This way it is also possible to write values in your own OD.

4.4.5 Function Block CAN_SDO_READ_BIN

FB for reading binary data from the Object Dictionary of a node via SDO transfer.

Prototype of the Function Block



Definition of Operands

DEVICE	Address of the node to be read (1-127 or 0 for local OD)
INDEX	Number of the index entry to be read
SUBINDEX	Number of the subindex entry to be read
SDOTYPE	Type of the SDO transfer mode to be used according to data type "CAN_SDO_TYPE" (see Table 11 in section 4.1.5) Note: If no value is assigned explicitly to this input (input open or not used), the function block uses mode "SDO_TYPE_AUTO_BEST_CASE". The network layer individually selects the best suitable SDO transfer mode based on the amount of data to be transferred.
PTR_RXDATA	Address of an object for receiving the read data bytes
MAXLENGTH	Limitation of number of bytes to read, if 0, the length of the object addressed by PTR_RXDATA is internally determined and used as the number of bytes to be read (the maximum of bytes reads equals the number of bytes the object is able to take up).
RXLENGTH	Number of read data bytes
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
ERRORINFO	SDO Abort Code of the communication partner according to data type "CIA405_SDO_ERROR" (see Table 12, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_SDO_READ_STR is used for reading binary data from the Object Dictionary of a node by using the SDO transfer. The SDO transfer always takes place in the background; the procedure described in section 4.1.3 therefore has to be used for synchronization between the function block and the PLC program, using the parameters ENABLE and CONFIRM.

If output CONFIRM has been set to TRUE when the function block returns, then the object addressed by element PTR_RXDATA contains the binary data bytes of the read object entry. Output RXLENGTH states the number of read bytes.

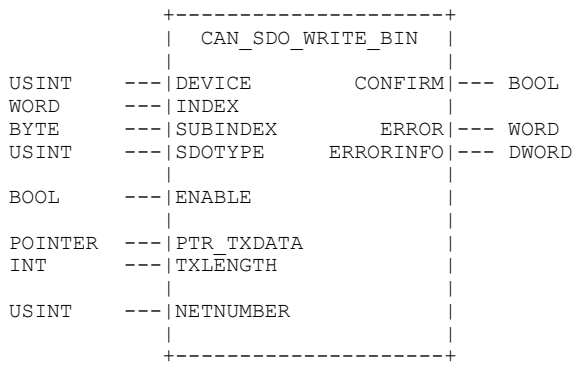
The network layer only supports a limited number of SDO transfers via the PLC program at a time (Standard: max. 5 transfers, if there is no deviating entry in the manual of the PLC). After starting the SDO transfer by setting ENABLE to TRUE, the respective SDO channel is locked and cannot be used by other blocks. The lock state remains until the SDO function block has been recalled with input ENABLE set to FALSE (see section 4.1.3). If calling through ENABLE = FALSE is skipped, the resource remains permanently locked and is no longer available for the PLC.

The local Object Dictionary of the PLC can be accessed by calling the function block with DEVICE = 0. This way it is also possible to read values from your own OD.

4.4.6 Function Block CAN_SDO_WRITE_BIN

FB for writing binary data into the Object Dictionary of a node via SDO transfer.

Prototype of the Function Block



Definition of Operands

DEVICE	Address of the node to be written (1-127 or 0 for local OD)
INDEX	Number of the index entry to be written
SUBINDEX	Number of the subindex entry to be written
SDOTYPE	Type of the SDO transfer mode to be used according to the data type "CAN_SDO_TYPE" (see Table 11, section 4.1.5) Note: If no value is explicitely assigned to this input (input open or not used), the function block uses mode "SDO_TYPE_AUTO_BEST_CASE". The network layer individually selects the best suitable SDO transfer mode based on the amount of data to be transferred.
PTR_TXDATA	Address of an object with binary data to be written
TXLENGTH	Number of bytes to write, if 0, the length of the object addressed by PTR_TXDATA is determined internally and used as the number of bytes to be written
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
ERRORINFO	SDO Abort Code of the communication partner according to data type "CIA405_SDO_ERROR" (see Table 12, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_SDO_WRITE_STR is used for writing binary data into the Object Dictionary of a node by using the SDO transfer. The SDO transfer always takes place in the background; the procedure described in section 4.1.3 therefore has to be used for synchronization between the function block and PLC program, using the parameters ENABLE and CONFIRM.

The address of an object with the binary data to be written in to the Object Dictionary has to be transferred to element PTR_TXDATA. Input TXLENGTH specifies the number of valid bytes. If this

value is 0, the length of the object addressed via PTR_TXDATA is internally determined and used as the number of bytes to be written.

The network layer only supports a limited number of SDO transfers via the PLC program at a time (Standard: max. 5 transfers, if there is no deviating entry in the manual of the PLC). After starting the SDO transfer by setting ENABLE to TRUE, the respective SDO channel is locked and cannot be used by other blocks. The lock state remains until the SDO function block has been recalled with input ENABLE set to FALSE (see section 4.1.3). If calling with ENABLE = FALSE is skipped, the resource remains permanently locked and is no longer available for the PLC.

The local Object Dictionary of the PLC can be accessed by calling the function block with DEVICE = 0. This way it is also possible to write values in your own OD.

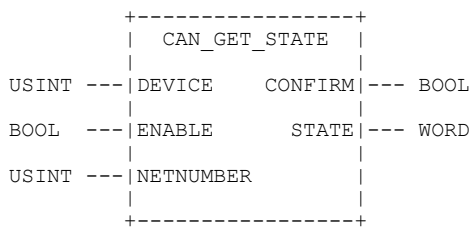
4.5 Function Blocks for Master Services

The function blocks for Master services enable state requests, monitoring and changing of the operational mode of any node, and the generation of SYNC messages. CANopen services are used which are exclusively permitted for just one node in the network (e.g. NMT services). Therefore, the functionality "PLC with CANopen Master" is connected to a specific node address so that only one control in the network can use this function (see section 4.1.2).

4.5.1 Function Block CAN_GET_STATE

FB for requesting the node state of devices.

Prototype of the Function Block



Definition of Operands

DEVICE	Address of the node to be requested (1-127 or 0 for own node)
STATE	Node state according to data type "CIA405_STATE" (see Table 9, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_GET_STATE is used to request the node state of a device. The state request is based on monitoring via Heartbeat or Lifeguarding. See section 2.2 for detailed information. The return values at output STATE have the following meaning:

UNKNOWN: The CANopen device at the specified address supports neither Heartbeat nor Lifeguarding, i.e. its state cannot be monitored. On a PLC without CANopen Master this state is also reported if there is no PLC with CANopen Master available in the network which supports state forwarding according to section 2.2 or if this Master PLC is in stop state (PLC program has stopped).

NOT_AVAIL: The CANopen device at the specified address no longer responds to Heartbeat or Lifeguarding requests and is, thus, no longer available for the system.

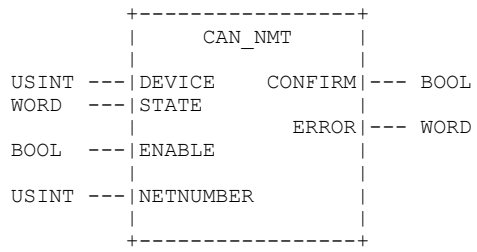
Other: Except for the state values UNKNOWN and NOT_AVAIL the return values correspond with the respective definitions in the CiA Draft Standard 301 (see Table 9, section 4.1.5).

The local node state of your own PLC is returned when calling the function block with DEVICE = 0.

4.5.2 Function Block CAN_NMT

FB for sending NMT messages.

Prototype of the Function Block



Definition of Operands

DEVICE Address of the node to be controlled (1-127 or 0 for all nodes)

STATE Node state according to data type "CIA405_TRANSITION_STATE" (see Table 10, section 4.1.5)

ERROR Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)

NETNUMBER Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)

ENABLE Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM Output for message completed through the FB (see section 4.1.3)

Description

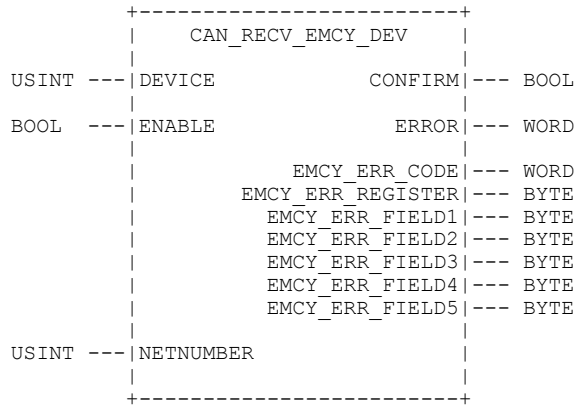
The function block CAN_NMT is used to control the state of a node (DEVICE = 1...127) or with DEVICE = 0 all nodes in the network. With the aid of this block a PLC program can also set the network to the Operational state manually if the CANopen Master has suppressed automatic network start. This is, e.g., the case if Bit 3 ("Do not send NMT-Start Remote Node") has been set in object 1F80H or if errors occurred during the configuration phase (see section 2.2). Table 8 contains the possible states (see section 4.1.5).

This function block is only available on a control in mode "PLC with CANopen Master" (see section 4.1.2).

4.5.3 Function Block CAN_RECV_EMCY_DEV

FB for reading emergency messages of a specific node from the receive buffer of the network layer.

Prototype of the Function Block



Definition of Operands

DEVICE — Address of the node (1-127) for which the reception of emergency messages is to be checked

EMCY_ERR_CODE
EMCY_ERR_REGISTER
EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5
Emergency error information according to CiA Draft Standard 301

ERROR — Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)

NETNUMBER — Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)

ENABLE — Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM — Output for message completed through the FB(see section 4.1.3)

Description

The function block CAN_RECV_EMCY_DEV is used for reading emergency messages of a specific node from the receive buffer of the network layer. If output CONFIRM has been set to TRUE when the function block returns, the elements EMCY_ERR contain the emergency error information of the node

according to the CiA Draft Standard 301. However, if output CONFIRM has been set to FALSE, the receive buffer of the network layer does not contain any emergency message for the respective node.

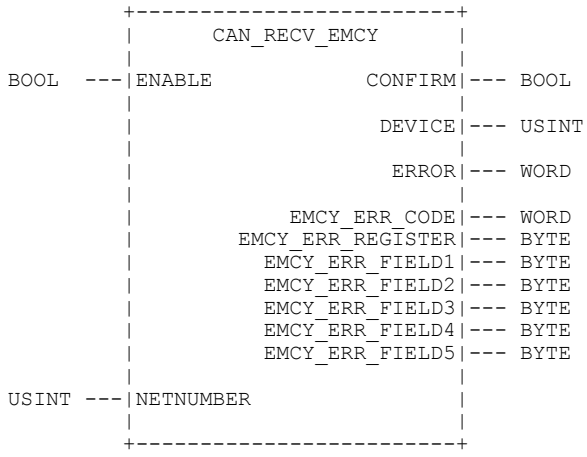
The function block always returns the emergency message of the respective node which has been entered first in the receive buffer (= oldest message), the message is subsequently deleted from the receive buffer. Each emergency message can thus only be read once by the PLC program. The function blocks CAN_RECV_EMCY_DEV and CAN_RECV_EMCY (see section 4.5.4) both access the same receive buffer.

This function block is only available on a control in mode "PLC with CANopen Master" (see section 4.1.2).

4.5.4 Function Block CAN_RECV_EMCY

FB for reading emergency messages of any node from the receive buffer of the network layer.

Prototype of the Function Block



Definition of Operands

DEVICE — Address of the node (1-127) which received an emergency message

EMCY_ERR_CODE
EMCY_ERR_REGISTER
EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5
Emergency error information according to CiA Draft Standard 301

ERROR — Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)

NETNUMBER — Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)

ENABLE — Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM — Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_RECV_EMCY is used for reading emergency messages of any node from the receive buffer of the network layer. If output CONFIRM has been set to TRUE when the function block returns, output DEVICE states the node address which received a message. The elements

EMCY_ERR contain the emergency error information of the node according to the CiA Draft Standard 301. However, if output CONFIRM has been set to FALSE, the receive buffer of the network layer does not contain any emergency message.

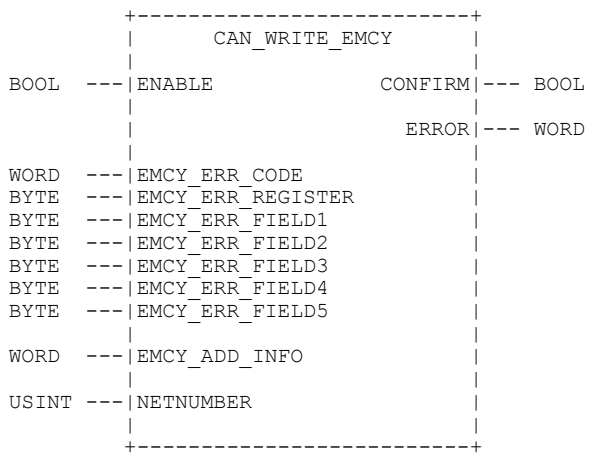
The function block always returns the emergency message of the respective node which has been entered first in the receive buffer (= oldest message), the message is subsequently deleted from the receive buffer. Each emergency message can thus only be read once by the PLC program. The function blocks CAN_RECV_EMCY_DEV (see section 4.5.3) and CAN_RECV_EMCY both access the same receive buffer.

This function block is only available on a control in mode "PLC with CANopen Master" (see section 4.1.2).

4.5.5 Function Block CAN_WRITE_EMCY

FB for sending application-specific emergency messages via the network layer.

Prototype of the Function Block



Definition of Operands

EMCY_ERR_CODE
EMCY_ERR_REGISTER
EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5
: Emergency error information according to the CiA Draft Standard 301 for the emergency message to be sent

EMCY_ADD_INFO
: Additional, user-specific emergency error information; is entered in Index 1003H of the Object Dictionary (Error Field, see CiA Draft Standard 301); is not a component of the emergency message to be sent, but is used for diagnosis purposes and can, therefore, also be zero

ERROR
: Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)

NETNUMBER
: Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)

ENABLE
: Input for enabling or disabling the FB (see section 4.1.3)

CONFIRM
: Output for message completed through the FB (see section 4.1.3)

Description

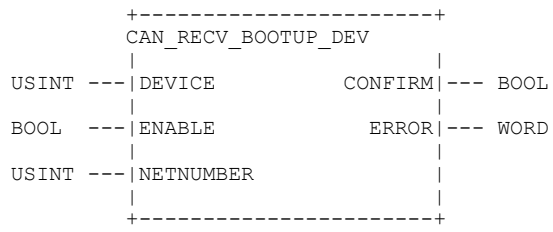
The function block CAN_WRITE_EMCY is used to send application-specific emergency messages via the network layer. The elements EMCY_ERR contain the emergency error information of the user application to be sent according to the CiA Draft Standard 301. Via element EMCY_ADD_INFO, the user application can transfer further error information which is not a component of the emergency message to be sent, but entered in Index 1003H of the Object Dictionary (Error Field, see CiA Draft Standard 301). This error information is only used for diagnosis purposes and can, therefore, also be zero. Index 1003H of the Object Dictionary can be read via a configuration or diagnosis tool.

When calling function block CAN_WRITE_EMCY, the message to be sent is stored in the send buffer of the CANopen kernel. If no error occurs during this phase (message could be successfully stored in the send buffer), the block whose output CONFIRM has been set to TRUE returns. However, there is no feedback to the PLC program stating whether the message has been sent successfully or not.

4.5.6 Function Block CAN_RECV_BOOTUP_DEV

FB for reading bootup messages of a specific node from the receive buffer of the network layer.

Prototype of the Function Block



Definition of Operands

DEVICE	Address of the node (1-127) for which the reception of bootup messages is to be checked
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_RECV_BOOTUP_DEV is used for reading bootup messages of a specific node from the receive buffer of the network layer. If output CONFIRM has been set to TRUE when the function block returns, the specified node received a bootup message. However, if output CONFIRM has been set to FALSE, the receive buffer in the network layer does not contain a bootup message for the respective node.

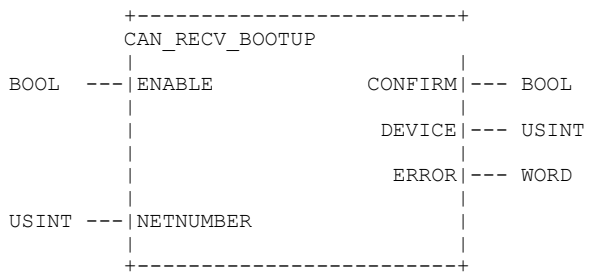
After reading a bootup message, it is deleted from the receive buffer and, therefore, only reported once to the PLC program. The function blocks CAN_RECV_BOOTUP_DEV and CAN_RECV_BOOTUP (see section 4.5.7) both access the same receive buffer.

This function block is only available on a control in mode "PLC with CANopen Master" (see section 4.1.2).

4.5.7 Function Block CAN_RECV_BOOTUP

FB for reading bootup messages of any node from the receive buffer of the network layer.

Prototype of the Function Block



Definition of Operands

DEVICE	Address of the node (1-127) which received a bootup message
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_RECV_BOOTUP is used for reading bootup messages of any node from the receive buffer of the network layer. If output CONFIRM has been set to TRUE when the function block returns, output DEVICE states the node address which received a message. However, if output CONFIRM has been set to FALSE, the receive buffer in the network layer does not contain bootup messages.

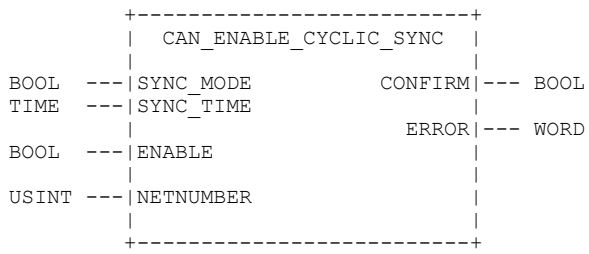
The function block always returns the bootup message of the respective node which has been entered first in the receive buffer (= oldest message); the message is subsequently deleted from the receive buffer. Each bootup message can thus only be read once by the PLC program. The function blocks CAN_RECV_BOOTUP_DEV (see section 4.5.6) and CAN_RECV_BOOTUP both access the same receive buffer.

This function block is only available on a control in mode "PLC with CANopen Master" (see section 4.1.2).

4.5.8 Function Block CAN_ENABLE_CYCLIC_SYNC

FB for activating or deactivating cyclic SYNC messages.

Prototype of the Function Block



Definition of Operands

SYNC_MODE	TRUE = Activate the generation of cyclic SYNC messages FALSE = Deactivate the generation of cyclic SYNC messages
SYNC_TIME	Time between two consecutive SYNC messages or 0 for SYNC message after each PLC cycle
ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_ENABLE_CYCLIC_SYNC is used for activating or deactivating the generation of cyclic SYNC messages via the PLC. When activated, the control generates a SYNC message between two consecutive PLC cycles if the last SYNC at least dates back to the time specified at input SYNC_TIME. If the time since the last SYNC is less than SYNC_TIME, no SYNC message is generated. The input value SYNC_TIME = 0 prompts the control to finish each PLC cycle with a SYNC message. In this case, the exchange of the network process image takes place synchronous to the exchange of the process image for the local inputs and outputs of the PLC.

At the end of each PLC cycle the control always checks the time specification for sending a SYNC message. The time specified at input SYNC_TIME is, therefore, the minimum time between two consecutive SYNC messages. The real time interval between two SYNC messages can, in the worst case scenario, vary by the length of one PLC cycle:

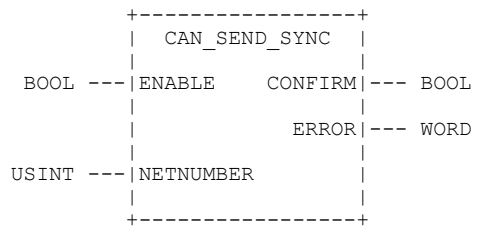
$T_{sync\ [worst\ case]}$ = SYNC_TIME + $T_{PLC\ cycle}$

This function block is only available on a control in mode "PLC with CANopen Master" (see section 4.1.2) and can only be used as an alternative to function block CAN_SEND_SYNC (see section 4.5.9).

4.5.9 Function Block CAN_SEND_SYNC

FB for sending an individual SYNC message.

Prototype of the Function Block



Definition of Operands

ERROR	Error code according to data type "CIA405_CANOPEN_KERNEL_ERROR" (see Table 8, section 4.1.5)
NETNUMBER	Network number (Note: If the PLC only supports one CANopen interface, setting of this input can be skipped since numeric variables have already been set with the initial value 0 according to IEC61131)
ENABLE	Input for enabling or disabling the FB (see section 4.1.3)
CONFIRM	Output for message completed through the FB (see section 4.1.3)

Description

The function block CAN_SEND_SYNC is used for generating individual SYNC messages with complete control of the PLC program. Each time the block with output ENABLE set to TRUE is called, a SYNC message is sent. Due to these targeted measures it is possible to only generate SYNC messages when really relevant data have been changed (e.g. in connection with the function blocks for PDOs and CAN Layer 2 messages CAN_PDO_READ8 or CAN_PDO_WRITE8, see section 4.3).

This function block is only available on a control in mode "PLC with CANopen Master" (see section 4.1.2) and can only be used as an alternative to function block CAN_ENABLE_CYCLIC_SYNC (see section 4.5.8).

4.6 Example Project for CANopen Function Blocks

The example project *"CopDemo"* included in the *OpenPCS* standard installation shows the application of various CANopen function blocks. It contains two programs with identical functionality, whereby one program realizes data exchange via PDO function blocks (*"PDODEMO.POE"*) and the other one via SDO function blocks (*"SDODEMO.POE"*).

Both demo programs of the example project *"CopDemo"* read process data from a CANopen IO module (node address 40H) and subsequently write it back onto the IO module. Therefore, the outputs of the IO module follow its input values. The demo program for the PDO function blocks reads the PDO generated by the IO module as a reaction to the changes at its inputs and returns the process data contained therein to the IO module via PDO. In the demo program for the SDO functionality, the PLC cyclically reads the current value of the inputs from the respective entry in the Object Dictionary of the node und subsequently writes it back onto the OD entry for the outputs via SDO.

The following consitheations are based on the demo program for SDO function blocks (*"SDODEMO.POE"*). The internal process control is based on the variable *ProcStep* which states the

respective current process step and is used for program linking. Its value is incremented after each successful completion of a process step:

```
(* === SDO processing === *)
RunCycle:
LD    ProcStep
EQ    1
JMPC  StartSDORead
LD    ProcStep
EQ    2
JMPC  RunSDORead
```

As described in section 4.1.3 (synchronization between CANopen function blocks and PLC program) a rising edge is required at input ENABLE to start an SDO transfer. To achieve this, instance *FB_SDO_Read8* is at first called with *ENABLE := FALSE*. During the subsequent second call *ENABLE := TRUE* is set and the required CANopen communication parameters are additionally transferred (*DEVICE*, *INDEX*, *SUBINDEX*):

```
(* Init SDO read *)
StartSDORead:
CAL   FB_SDO_Read8 (
      ENABLE := FALSE)

CAL   FB_SDO_Read8 (
      DEVICE := RX_DEVICE,
      INDEX := RxIndex,
      SUBINDEX := RxSubIndex,
      ENABLE := TRUE)

LD    ProcStep
ADD   1
ST    ProcStep
RET
```

After starting the SDO transfer, the variable *ProcStep* used for the internal process control is incremented so that the process step for waiting for completion of the SDO transfer is called from the next PLC cycle onwards. The PLC repeats this process step cyclically as long as the output value of *FB_SDO_Read8.CONFIRM* is set to FALSE and output *FB_SDO_Read8.ERROR* does not display errors (equals zero):

```
(* Processing SDO read *)
RunSDORead:
CAL   FB_SDO_Read8 (
      ENABLE := TRUE)

LD    FB_SDO_Read8.CONFIRM
JMPC  RunSDOReadEnd
LD    FB_SDO_Read8.ERROR
EQ    0
RETC

RunSDOReatheror:
LD    ProcStep  (* restart SDO transfer after error *)
SUB   1
ST    ProcStep
RET
```

If the function block returns to *FB_SDO_Read8.ERROR* with an output value unequal to zero, the SDO transfer has been cancelled due to an error. The error code of *FB_SDO_Read8.ERROR* then states the reason (according to Table 8, section 4.1.5). As a result of this, the value of the variable *ProcStep* is decremented again to ensure that the process step for starting the SDO transfer is executed again in the following PLC cycle.

If output *FB_SDO_Read8.CONFIRM* contains the value TRUE, the SDO transfer has been successful. In this case, program execution is continued at label *RunSDOReadEnd* (see below). As described in section 4.1.3 (synchronization between CANopen function block and PLC program), the PLC program has to inform the network layer about completion of the SDO transfer. To achieve this, instance *FB_SDO_Read8* is recalled with *ENABLE := FALSE*. This also causes the network layer to reactivate the SDO channel so that it can be used by other function blocks again. The read data is subsequently transferred locally and the variable *ProcStep* is re-incremented. The SDO transfer is now completely finished; from the next PLC cycle, the control continues its program execution with the following process step:

```
RunSDOReadEnd:
CAL    FB_SDO_Read8 (
       ENABLE := FALSE)

       (* copy received data *)
LD     FB_SDO_Read8.DATALENGTH
ST     DataLength
LD     FB_SDO_Read8.DATA0
ST     Data[0]

LD     ProcStep
ADD    1
ST     ProcStep
RET
```

Without the last call *ENABLE := FALSE* the SDO channel remains in the lock state and can only be used by the locking instance – in this case *FB_SDO_Read8*. For the demo program *"SDODEMO.POE"* this would mean that the subsequent call of instance *FB_SDO_Write8* for returning the data (not described here) aborts with the error message TRANSFER_BUSY at output ERROR. Sending of the data would no longer be possible.

5 Configuration of a PLC with CANopen Master

5.1 Basic Information for the Master Configuration

The basis for the configuration of the CANopen Master is the DCF file of the Master PLC. Integration of the DCF file into the PLC configuration is described in section 3.3. The configuration objects described below have to be parameterized accordingly in this DCF file. It is recommended to use the "SYSTEC CANopen Master Configurator" which is opened in the *OpenPCS* programming environment via the menu item "Extras -> Tools -> CANopen Master Configurator" (see Figure 17). The Master Configurator is described in detail in the manual L-1109 "Software Manual CANopen Master Configurator".

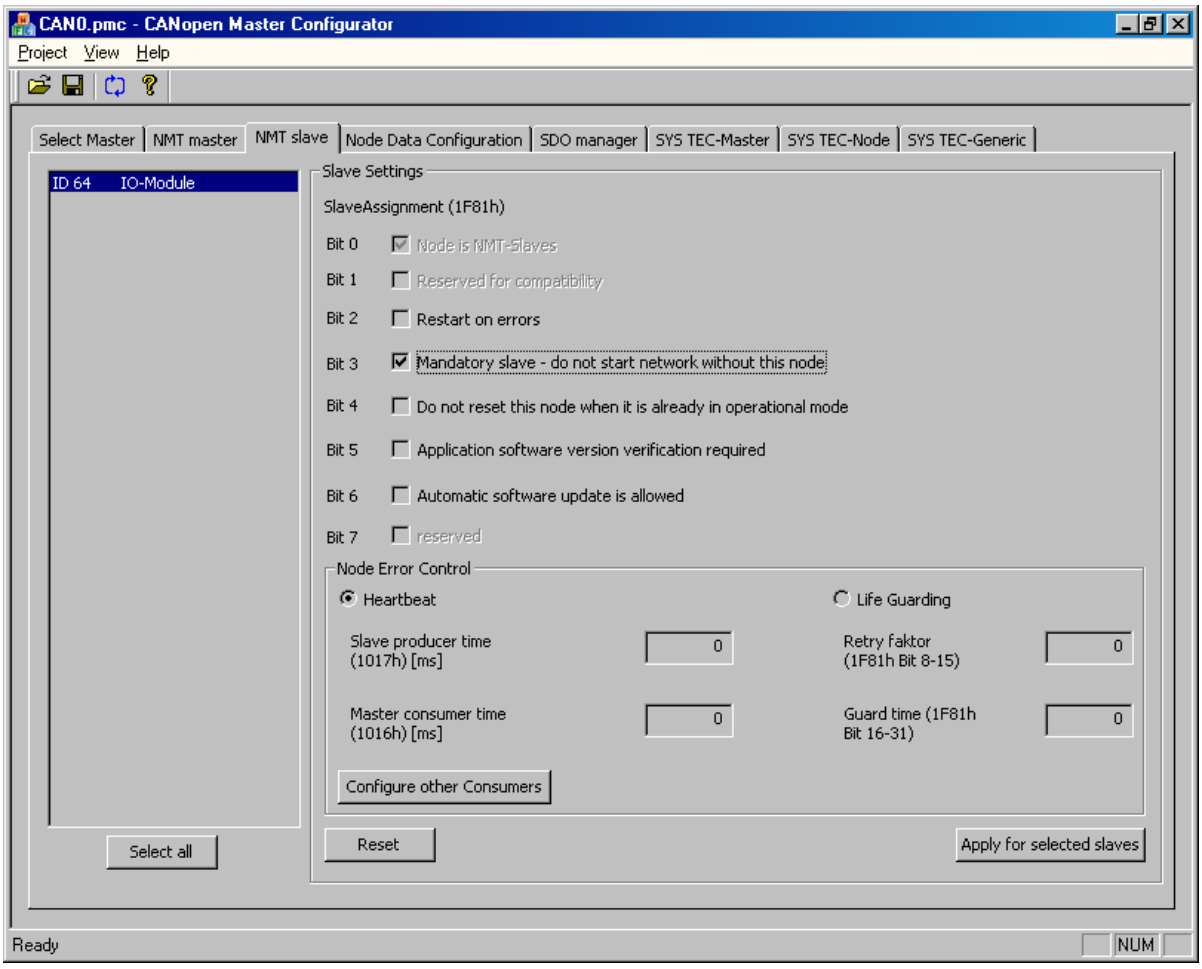


Figure 17: SYSTEC CANopen Master Configurator

5.2 Definition of the Node List

Definition of the nodes to be configured and monitored by the CANopen Master takes place via the node list in the object entry 1F81H (NMT Slave Assignment, see CiA DS302 V3). Configuration of this object should preferably be carried out in the "NMT Slave" view of the "SYSTEC CANopen Master Configurator".

If object 1F81H has not been created in the Object Dictionary of the Master PLC (i.e. object 1F81H is not contained in the DCF file of the Master PLC), the control determines all the CANopen devices available in the network via a network scan which, as standard, covers the entire range for all node addresses from 1 to 127. Due to the application-specific requirements it might be necessary to limit the scanning range to ensure that not all the available node numbers between 1 and 127 are included in the network scan. A manufacturer-specific object with the Index 3001H is created for setting interval limits. This object contains two sub-entries for marking the lower limit (Subindex 1) and the upper limit (Subindex 2). See Table 13 for a description of the design of object 3001H. The network scan then only takes place within this interval (incl. lower and upper limit). Node monitoring for nodes with a node number outside this interval is not possible.

Table 13: Object Dictionary entry for interval limits of the network scan

Index	Sub-index	Name	Default Value	Type	Attr.	Meaning
3001H	0	Number of entries	2	u8	ro	
	1	Network scan Upper limit	1	u8	rw	First valid node number for network scan
	2	Network scan Lower limit	127	u8	rw	Last valid node number for network scan

If object 3001H is not in the Object Dictionary, the specified default values apply.

5.3 Configuration of the COBID for Node State Messages

Basis for the CANopen function block *CAN_GET_STATE,* with which a PLC program can determine the current state of a network node, is node monitoring via Heartbeat or Lifeguarding. Since this node monitoring only takes place on the Master PLC, the current states of the network nodes are unknown to all the other controls. Therefore, a Master PLC sends a broadcast message with the current node state to all controls without CANopen Master when recognizing a change of a node state. This ensures that the new state is also known on these controls and can be requested there by calling the CANopen function block *CAN_GET_STATE*. The default COBID can be configured via the Object Dictionary to prevent conflicts during integration of third party CANopen devices (see Table 14). When changing the default COBID, please note that this has to be carried out for all nodes concerned.

Table 14: Object Dictionary entry for COBID of the node state messages

Index	Sub-index	Name	Default Value	Type	Attr.	Meaning
3002H	0	Number of entries	3	u8	ro	
	1	COBID node state message	50H	u32	rw	COBID of the node state message which is sent by the Master PLC when a monitored node changes its state

If object 3002H is not in the Object Dictionary, the specified default values apply.

5.4 Definition of Waiting Periods

When starting the PLC program, the Master PLC sends the NMT command "Enter Pre-Operational State", followed by "Start Remote Node" for all nodes (node address = 0). This prompts the CANopen nodes to send their PDOs once. Subsequently, the PLC waits until the PDOs have been processed and the received values have been stored in the network image before starting the PLC program. This ensures the initial initialization of the network variables. The respective waiting periods can be configured via object 3003H in the Object Dictionary of the Master PLC (see Table 15).

Table 15: Object Dictionary entries for defining waiting periods

Index	Sub-index	Name	Default Value	Type	Attr.	Meaning
3003H	0	Number of entries	1	u8	ro	
	1	Boot Network Delay	50	u16	rw	Waiting period in [ms] between sending the NMT commands *EnterPreOperational* and *EnterOperational* when starting the PLC program
	2	PLC Start Delay	100	U16	rw	Waiting period in [ms] between the NMT command *EnterOperational* and the start of the PLC program in order to process the initial PDOs and to store the received values in the network image

If object 3003H is not in the Object Dictionary, the specified default values apply.

5.5 Configuration of Heartbeat/Lifeguarding of the CANopen Devices

The CANopen Master uses Heartbeat or Lifeguarding to monitor the CANopen devices. Monitoring of the node is primarily determined by the respective entries in the DCF file of the respective node. If there is no DCF file available for the respective node, selection of the monitoring method as well as configuration of the monitoring time occurs via the DCF file of the PLC (which has to be operated in Master mode). If neither Heartbeat nor Lifeguarding has been configured as monitoring methods in the DCF file, the Master uses the optimum method (i.e. Heartbeat). If Heartbeat is not available for the device, Lifeguarding is used.

Configuration of Heartbeat:

Configuration of Heartbeat occurs with object entry 1016H (Heartbeat Consumer, also see CiA DS301 V4.x). A free subindex entry has to be carried out for each device to be monitored. The subindex can be selected freely. The entry has the following structure.

Data type: UNSIGNED32

Table 16: Configuration of Heartbeat

Bits	31-24	23-16	15-0
Value	Reserved (00H)	Node-ID	Heartbeat Time
Encoded as	-	UNSIGNED8	UNSIGNED16
Example, Subindex 1	00h	40H	0BB8H

In the example, the Heartbeat Consumer is set to 3000ms for the device with the node address 40H. The Heartbeat Producer of the device is configured with a third of the value (here 1000ms). This means that the Master PLC stores the value 1000 in the object entry 1017H for the device.

See Figure 18 for details of the Heartbeat configuration process.

Configuration of Lifeguarding:

Configuration of Lifeguarding occurs with the object entry 1F81H (NMT Slave Assignmet, see CiA DS302 V3). For each device to be monitored, the entry has to be carried out in the subindex which corresponds to the node address of the device. The entry has the following structure:

Data type: UNSIGNED32

Table 17: Configuration of Lifeguarding

Bits	**31-16**	**15-8**	**7-0**
Value	Guard Time	Retry Factor	Not used
Encoded as	UNSIGNED16	UNSIGNED8	UNSIGNED8
Example, Subindex 41H	05DCH	41H	0H

In the example, Lifeguarding is set with a time of 1500ms for the device with node address 41H. The repetition factor for the device is set to 5. The Master PLC sets the following values in the device:

Guard Time	Object entry 100CH:	1500ms
Lifetime Factor	Object entry 100DH:	5

The interval between two requests of the Master equals the Guard Time multiplied by a safety factor of 0.8. In this example, a request is carried out approx. every 1200ms.

See Figure 19 for details of the Lifeguarding configuration process.

Configuration with Default Values:

If neither Heartbeat nor Lifeguarding has been configured for the CANopen device in the DCF file, the Master PLC uses default values for node monitoring. The configuration takes place as described below:

1. Does the device support Heartbeat?
   Access to the object entry 1017H of the device: If the object entry is available, the Heartbeat Producer of the device (object 1017H) is set to 1000ms and the Heartbeat Consumer of the Master PLC (object 1016H) is set to 3000ms.

2. If the device supports Lifeguarding instead of Heartbeat, a lifetime (object 100CH) of 1000ms and a lifetime factor (object 100DH) of 3 are set in the device. These parameters are stored for the device in the Master PLC in the object entry 1F81H.

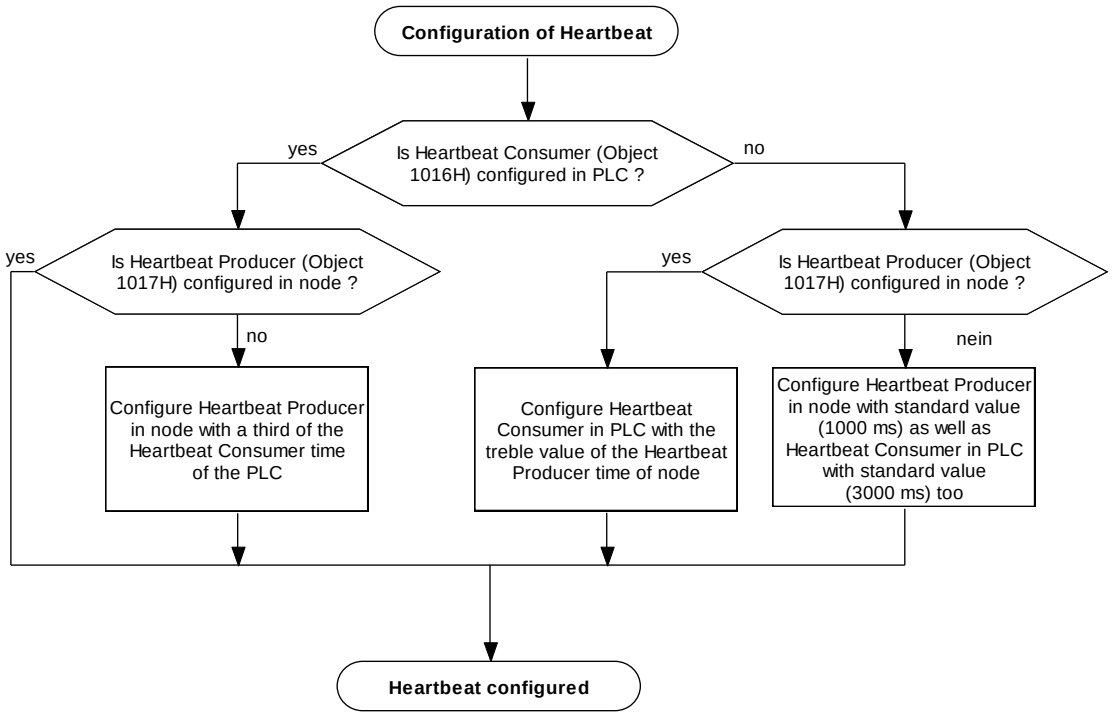


Figure 18: Heartbeat configuration process

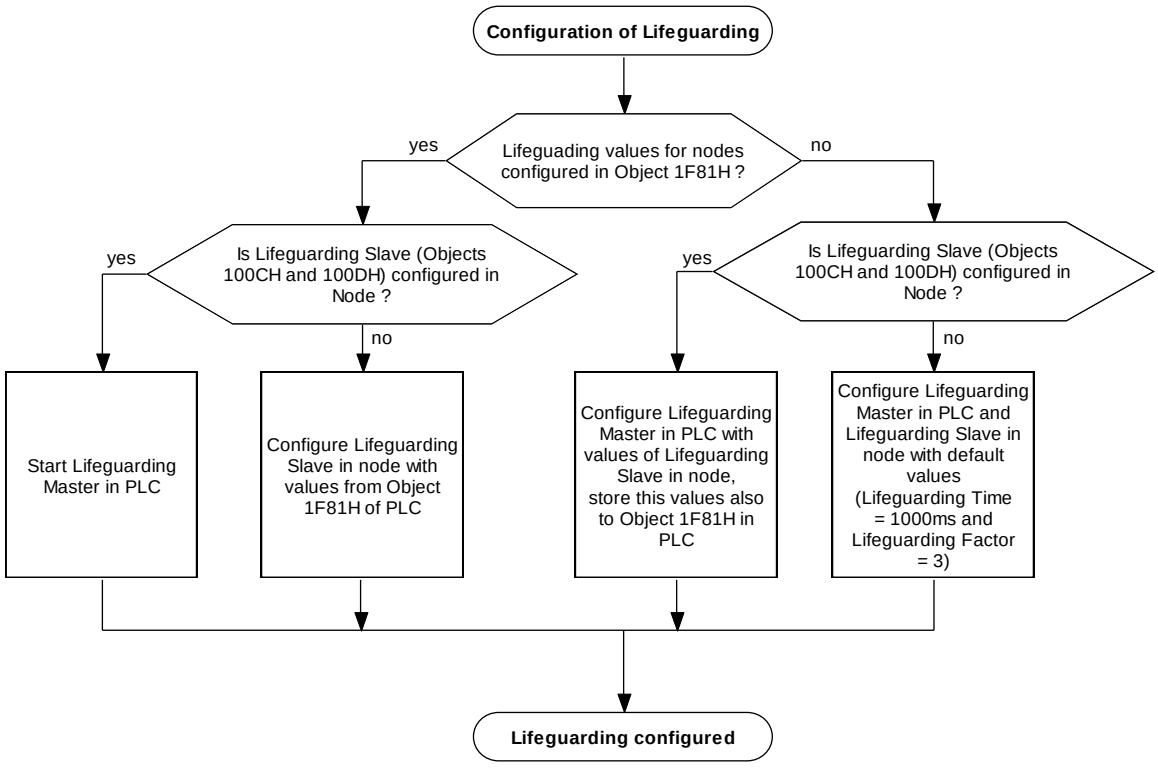


Figure 19: Lifeguarding configuration process

Part 2

CAN Layer 2

(Low Level Protocol)

6 IEC61131 Function Blocks for CAN Layer 2

6.1 Basic Information on CAN Layer 2 Function Blocks

Within the IEC61131-3 different manufacturer-specific function blocks are available for a direct access to the CAN-interface. Herewith, CAN-telegrams can be written and processed directly through the PLC program. Furthermore, those function blocks support CAN-messages in the Extended-Frame Format (29 Bit CAN identifier according to CAN 2.0B). The CAN Layer 2 function blocks allow the PLC a flexible data exchange with any devices, which are not CANopen capable.

Note: **CAN Layer 2 function blocks cannot be used simultaneously with CANopen services on the same CAN-Interface. CAN Layer 2 function blocks can only be used if the CANopen-functionality for the respective CAN-Interface has been disabled in advance. (see section 6.2.1)**

6.1.1 Overview of CAN Layer 2 Funtion Blocks

Table 18 shows an overview about the CAN Layer 2 function blocks for the IEC61131-3. All function blocks are realized as firmware blocks and hence part of the PLC firmware. Depending on the CAN controller inside the PLC are not all functionalities available in all cases. This especially concerns the filtering of CAN-messages (REGISTER_CANID / UNREGISTER_CANID) as well as the use of RTR-Frames. Availability or restrictions of particular functionalities are listed in the System Manual of the respective control.

Table 18: Overview of CAN Layer 2 funtion blocks for IEC 61131-3

Function Block	Meaning	Section
CANL2_INIT	Initializing of the CAN-Interface (Requires deactivation of the CANopen functionality for this interface)	6.2.1
CANL2_SHUTDOWN	Deactivation of the CAN-Interface	6.2.2
CANL2_RESET	Reset of the CAN-Interface	6.2.3
CANL2_GET_STATUS	Status request of the CAN-Interface	6.2.4
CANL2_DEFINE_CANID	Definition of a CAN-Object for data transfer	6.2.5
CANL2_DEFINE_CANID_RANGE	Definition of an range of CAN-Objects for data exchange	6.2.6
CANL2_UNDEFINE_CANID	Rejection of a previously defined CAN-object	6.2.7
CANL2_UNDEFINE_CANID_RANGE	Rejection of a previously defined range of CAN-objects	6.2.8
CANL2_MESSAGE_READ8	Reading out of received CAN-messages, data are passed at FB-outputs	6.2.9
CANL2_MESSAGE_READ_BIN	Reading out of received CAN-messages, data are written in the object addressed via pointer	6.2.10
CANL2_MESSAGE_WRITE8	Sending of CAN-massages, data are passed on FB-inputs	6.2.11
CANL2_MESSAGE_WRITE_BIN	Sending of CAN-messages, data are read from the object addressed via pointer	6.2.12
CANL2_MESSAGE_UPDATE8	Update of data of an RTR-message, data are passed on FB-inputs	6.2.13
CANL2_MESSAGE_UPDATE_BIN	Update of data of an RTR-message, data are read from the object addressed via pointer	6.2.14

6.1.2 Synchronisation Between CAN Layer 2 Function Block and PLC Program

The process-synchronisation between CAN-Interface and the PLC program occurs by means of signal ENABLE and CONFIRM of the function blocks. The meaning of the signals ENABLE and CONFIRM as well as the process of synchronisation are identical to the synchronisation described for CANopen function blocks in section 4.1.3 (for details refer to section 4.1.3).

6.1.3 CAN Layer 2-specific Constants

To mark error status, the data type "CANL2_ERROR" is used. Here, the error status which can occur within the CAN-Interface of an PLC, are summarized. These error codes are used by different function blocks as output parameter ERROR. Table 19 shows the allocation of used numeric constants to the respective error codes.

Table 19: Constants for data type "CANL2_ERROR"

Constant		Errorcode
16#00	(= 00 dez)	NO_ERROR
16#01	(= 01 dez)	OTHER_ERROR
16#02	(= 02 dez)	INVALID_NETNUMBER
16#03	(= 03 dez)	INVALID_PARAMETER
16#04	(= 04 dez)	NO_MESSAGE
16#05	(= 05 dez)	UNSUPPORTED_BITRATE
16#06	(= 06 dez)	INIT_FAILED
16#07	(= 07 dez)	DEVICE_BUSY
16#08	(= 08 dez)	TX_BUFFER_OVERRUN
16#09	(= 09 dez)	NO_FREE_CHANNEL
16#0A	(= 10 dez)	COBID_ALREADY_REGISTERED
16#0B	(= 11 dez)	POINTER_TYPE_NOT_SUPPORTED

The data type "CANL2_BUS_STATUS" is used for marking the status, in which a CAN-Interface is arranged. Table 20 shows the allocation of used numerical constants to the respective status values.

Table 20: Constants for Data Type "CANL2_BUS_STATUS"

Constant		Status value
16#00	(= 00 dez)	BUS_STATE_OK
16#01	(= 01 dez)	BUS_STATE_WARNING_LIMIT
16#02	(= 02 dez)	BUS_STATE_ERROR_PASSIVE
16#03	(= 03 dez)	BUS_STATE_BUS_OFF

The data type "CANL2_CDRV_STATUS" is used for signalling error and status states within the CAN-driver. Thereby, an own bit is assigned to each event within the status mask. Several events can occur parallel at a particular time. In this case, multiple bits are set simultaneously within the status mask (OR combination). Table 21 shows the allocation of used numeric constants to the respective status values.

Table 21: Constants for Data Type "CANL2_CDRV_STATUS"

Konstante		Statuswert
16#0000		CDRV_STATE_OK
16#0001	(Bit 0)	CDRV_STATE_WARNING_LIMIT_SET

16#0002	(Bit 1)	CDRV_STATE_WARNING_LIMIT_RESET
16#0004	(Bit 2)	CDRV_STATE_ERROR_PASSIVE_SET
16#0008	(Bit 3)	CDRV_STATE_ERROR_PASSIVE_RESET
16#0010	(Bit 4)	CDRV_STATE_BUS_OFF
16#0020	(Bit 5)	CDRV_STATE_OVERRUN (=Hardware Overrun)
16#0040	(Bit 6)	CDRV_STATE_STUFF_ERROR
16#0080	(Bit 7)	CDRV_STATE_FORM_ERROR
16#0100	(Bit 8)	CDRV_STATE_ACK_ERROR
16#0200	(Bit 9)	CDRV_STATE_CRC_ERROR
16#0400	(Bit 10)	CDRV_STATE_RX_BUFF_HIGH_OVERRUN
16#0800	(Bit 11)	CDRV_STATE_RESERVE1
16#1000	(Bit 12)	CDRV_STATE_RX_BUFF_LOW_OVERRUN
16#2000	(Bit 13)	CDRV_STATE_RESERVE2
16#4000	(Bit 14)	CDRV_STATE_BUS_OFF_RESET
16#8000	(Bit 15)	CDRV_STATE_FIRST_BUS_CONTACT

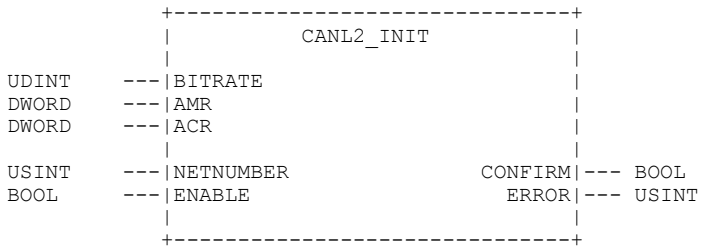
6.2 Function Blocks for CAN Layer 2

The function blocks for access to the local CAN open-kernel of the own PLC allow for a query of the node address as well as the status of the network layer. These function blocks do not require any communication with other nodes.

6.2.1 Function Block CANL2_INIT

FB for Initialization of the CAN-Interface.

Prototype of the function block



Definition of Operands

BITRATE Bitrate in Bit/s, e.g.:
BITRATE := 125000 125 kBit/s
BITRATE := 250000 250 kBit/s
BITRATE := 1000000 1 MBit/s

AMR Configuration value for Acceptance Mask Register (AMR) of the CAN-Controllers; allows for hardware filtration of CAN-Identifiers in the CAN-Controller (Standard value for processing of all CAN-Messages: AMR := 16#FFFFFFFF)

ACR Configuration Value for Acceptance Code Register (ACR) of the CAN-Controllers, allows for a hardware filtration of CAN-identifiers in the CAN-Controller (Standard value for processing of all CAN-Messages: ACR := 16#00000000)

NETNUMBER Network number

ERROR	Errorcode relates to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

Description

The function block CANL2_INIT is used for initializing the CAN-Interface. **This requires the deactivation of the CANopen functionality for this interface.** The procedure herefor is described in the system manual of the respective control. If the PLC supports the configuration through a WEB-Frontend, usually the *"Enable State"* of the interface concerned can be set to *"Disabled".* Thereby, the interface is not initialized automatically for CANopen and is available for free use through the PLC-program.

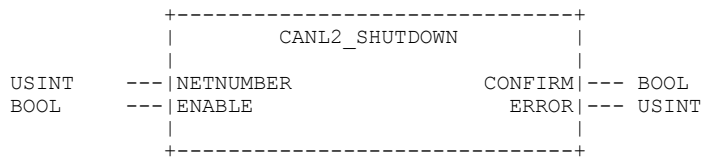
At the input BITRATE, the value for the Bitrate has to be specified in Bit/s,e.g. *125000* for 125 kBit/s. The values of both inputs AMR (Acceptance Mask Register) and ACR (Acceptance Code Register) are immediately transferred to the AMR or ACR register of the CAN-Controller. The result of a combination of both values is a receive filter, which can only be passed by CAN-messages whose CAN-Identifiers observe the filter criteria. Through appropriate settings of AMR and ACR, all non-relevant CAN-messages can be excluded from reception already in the CAN-Controller that, in turn, reduces the CPU-load of the control and can prevent an overflow of the receive buffer. The exact relevance of AMR and ACR as well as the realized filter values therewith can be taken from the data sheet of the respective CAN-Controller. Normally, the filters are set in a way the CAN-Controller receives all CAN-messages. For this purpose, AMR and ACR have to be reserved as follows:

```
AMR := 16#FFFFFFFF;
ACR := 16#00000000;
```

6.2.2 Function Block CANL2_SHUTDOWN

FB for Deactivation of the CAN-Interface

Prototype of the function block



Definition of Operands

NETNUMBER	Network number
ERROR	Errorcode relates to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

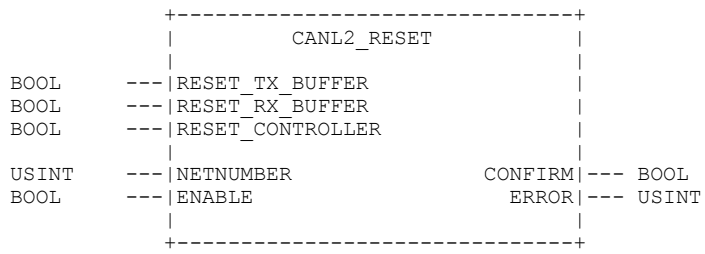
Description

The function block CANL2_SHUTDOWN serves for the deactivation of the CAN-Interface.

6.2.3 Function Block CANL2_RESET

FB to reset the the CAN-Interface

Prototype of the function block



Definition of Operands

RESET_TX_BUFFER	TRUE = Reset send buffer (reject all messages) FALSE = Do not change send buffer
RESET_RX_BUFFER	TRUE = Reset receive buffer (reject all messages) FALSE = Do not change receive buffer
RESET_CONTROLLER	TRUE = Reset CAN-Controller and initialize again FALSE= Do not change CAN-Controller
NETNUMBER	Network number
ERROR	Errorcode relates to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling of the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

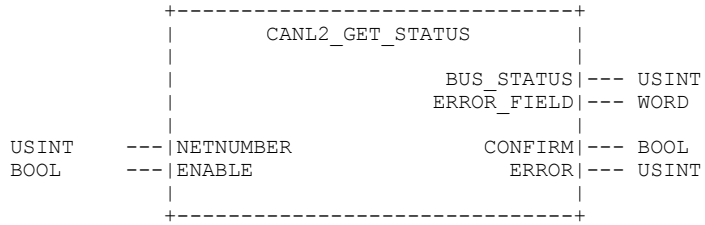
Description

The function block CANL2_RESET serves for reset of the CAN-Interface. Through the inputs RESET_TX_BUFFER, RESET_RX_BUFFER and RESET_CONTROLLER, the PLC program is able to control which parts of the interface are to reset. If the inputs RESET_TX_BUFFER, RESET_RX_BUFFER are set to TRUE, the respective bufferis reset and thereby, all messages still in the buffer are rejected. If the input RESET_CONTROLLER is set to TRUE, this results in the reset of the CAN-Controllers with a subsequent reinitialization.

6.2.4 Function Block CANL2_GET_STATUS

FB for status request of the CAN-Interface

Prototype of the function block



Definition of Operands

BUS_STATUS	Bus-Status according to data type "CANL2_BUS_STATUS" (see Table 20, section 6.1.3)
ERROR_FIELD	CAN-Driver-Status according to data type "CANL2_CDRV_STATUS" (see Table 21, section 6.1.3)
NETNUMBER	Network number
ERROR	Errorcode according to data type “CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

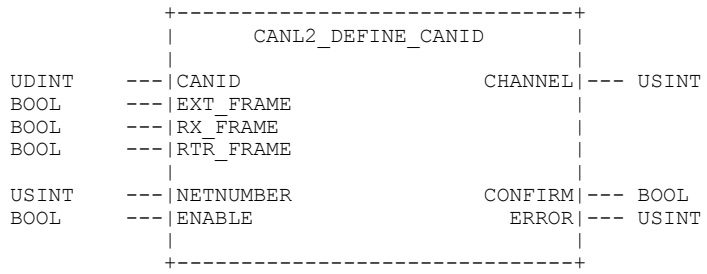
Description

The function block CANL2_GET_STATUS serves for querying the status of the CAN-Interface. The output BUS_STATUS provides information on the Bus-Status of the control.The output ERROR_FIELD informs about current status events within the CAN-driver. Thereby, an own bit is assigned to each event within the status mask at the output ERROR_FIELD. Several events can occur in parallel in one point in time; in this case, several bits are set simultaneously within the status mask (or-linking).

6.2.5 Function Block CANL2_DEFINE_CANID

FB for the definition of a CAN-Object for data exchange

Prototype of the function block



Definition of Operands

CANID	CAN-Identifier, the CAN-object has to be defined for
EXT_FRAME	TRUE = the CAN-Identifier marks an Extended-Frame (29 Bit) FALSE = the CAN-Identifier marks a Standard-Frame (11 Bit)
RX_FRAME	TRUE = the CAN-Object is used for receiving FALSE = the CAN-Object is used for sending
RTR_FRAME	TRUE = CAN-Object is used for RTR FALSE = CAN-Object is not used for RTR
CHANNEL	Returns the channel number for the CAN-Object assigned by the CAN-driver
NETNUMBER	Network number
ERROR	Error code according to data type "CANL2_ERROR" (see Table 19, section 6.1.3)

ENABLE	Input for enabling or disabling the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

Description

The function block CANL2_DEFINE_CANID serves for the definition of a CAN-Object for data exchange. The input EXT_FRAME indicates, if the CAN-Identifier, which has been passed at the input CANID, can be construed as an 11-Bit Identifier of a standard-message (CAN 2.0A) or as 29-Bit Identifier of an extended-message (CAN 2.0B). The input RX_FRAME defines, whether the object is used for either sending or receiving of data.

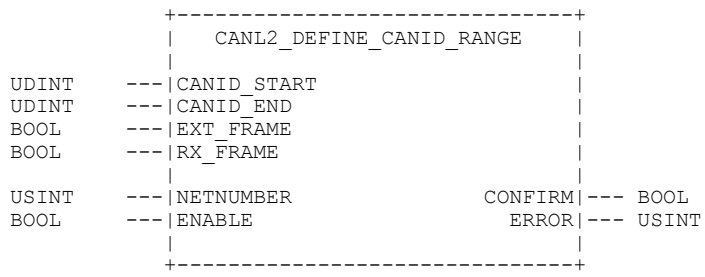
Through the input RTR_FRAME, a CAN-object can be marked as an RTR-object (Remote Transmission Request). Remote-Frames are only sent by a node if this node has been requested by another node.

The output CHANNEL delivers the channel number for the CAN-Object, which has been assigned by the CAN-driver. Especially for RTR-Objects, the channel number is of high meaning as principally each message has to be managed within an own channel of the CAN-Controller.

6.2.6 Function Block CANL2_DEFINE_CANID_RANGE

FB for defining CAN-Objects for data transfer

Prototype of the function block



Definition of Operands

CANID_START	first (lowest) CAN-Identifier for the range of CAN-objects to be defined
CANID_END	last (largest) CAN-Identifier for the range of CAN-objects to be defined
EXT_FRAME	TRUE = the CAN-Identifier-Range marks Extended-Frames (29 Bit) FALSE = the CAN-Identifier-Range marks Standard-Frames (11 Bit)
RX_FRAME	TRUE = the CAN-Objects are used for receiving FALSE = the CAN-Objects are used for sending
NETNUMBER	Network number
ERROR	Errorcode relates to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

Description

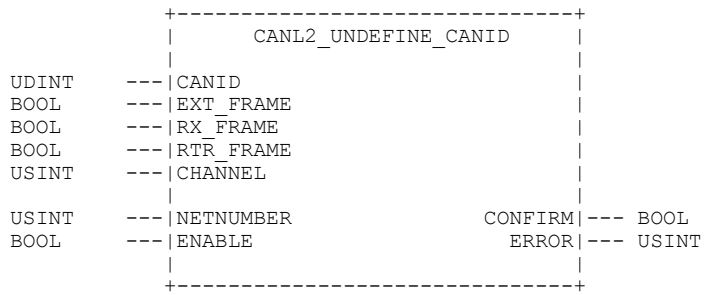
The function block CANL2_DEFINE_CANID_RANGE serves for the definition of a range of CAN-objects for data exchange. The input EXT_FRAME indicates, whether the CAN-Identifier range that has been defined through the inputs CANID_START and CANID_END, can be construed as 11-Bit Identifier for standard-messages (CAN 2.0A) or as 29-Bit Identifier for extended-messages (CAN 2.0B). The input RX_FRAME defines, whether the objects are used for either sending or receiving of data.

By means of function block CANL2_DEFINE_CANID_RANGE, no RTR-objects (Remote Transmission Request) can be designed. Therefor, function block CANL2_DEFINE_CANID has to be used and selected several times if needed.

6.2.7 Function Block CANL2_UNDEFINE_CANID

FB to reject a previously defined CAN-Object.

Prototype of the function block



Definition of Operands

CANID EXT_FRAME RX_FRAME RTR_FRAME	Here, the same parameters that have been used previously are to specify for the definition of the CAN-object by means of function block CANL2_DEFINE_CANID
CHANNEL	The channel number returned by function block CANL2_DEFINE_CANID
NETNUMBER	Network number
ERROR	Errorcode according to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE CONFIRM	Input for enabling or disabling the FB (see section 6.1.2) Output for message completed through the FB (see section 6.1.2)

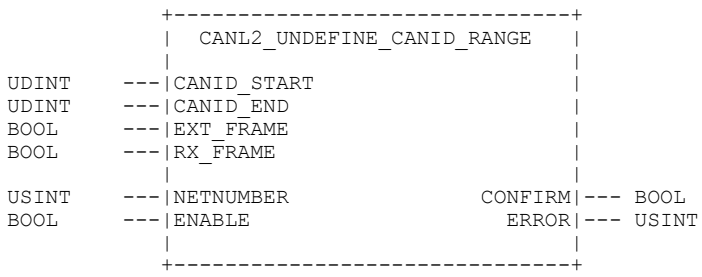
Description

The function block CANL2_UNDEFINE_CANID serves for the rejection of a CAN-object defined previously by means of function block CANL2_DEFINE_CANID. To clearly identify the CAN-object to be rejected, the same parameters used previously for the definition of the CAN-object by means of function block CANL2_DEFINE_CANID have to be specified at the inputs CANID, EXT_FRAME, RX_FRAME und RTR_FRAME.

6.2.8 Function Block CANL2_UNDEFINE_CANID_RANGE

FB for rejection of a previously defined range of CAN-objects

Prototype of the function block



Definition of Operands

CANID_START CANID_END EXT_FRAME RX_FRAME	Here, the same parameters used previously have to be specified to define the range of CAN-objects by means of the function blocks CANL2_DEFINE_CANID_RANGE
NETNUMBER	Network number
ERROR	Errorcode according to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE CONFIRM	Input for enabling or disabling the FB (see section 6.1.2) Output for message completed through the FB (see section 6.1.2)

Description

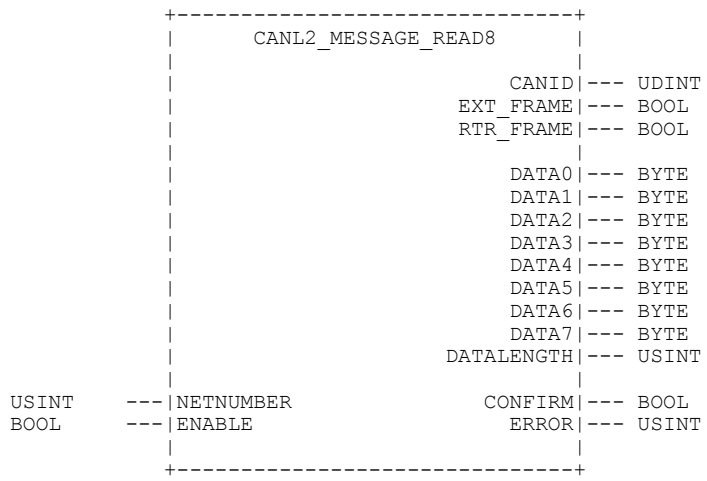
The function block CANL2_UNDEFINE_CANID_RANGE serves for rejection of a previously defined range of CAN-objects by means of function block CANL2_DEFINE_CANID RANGE.

To identify the range of CAN-objects to reject, the same parameters used for the definition of the CAN-object by means of function block CANL2_DEFINE_CANID_RANGE CANID_START, CANID_END, EXT_FRAME and RX_FRAME are to specify.

6.2.9 Function Block CANL2_MESSAGE_READ8

FB to read received CAN-messages (transfer of data at FB-outputs).

Prototype of the function block



Definition of Operands

CANID	CAN-Identifier of the received CAN-Message
EXT_FRAME	TRUE = the CAN-Identifier marks an Extended-Frame (29 Bit) FALSE = the CAN-Identifier marks a Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Object has been passed as RTR-Frame FALSE = CAN-Object has not been passed as RTR-Frame
DATA0 - DATA7	Data bytes of the received CAN-Message
DATALENGTH	Length of the received CAN-Message
NETNUMBER	Network number
ERROR	Errorcode according to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling of the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

Description

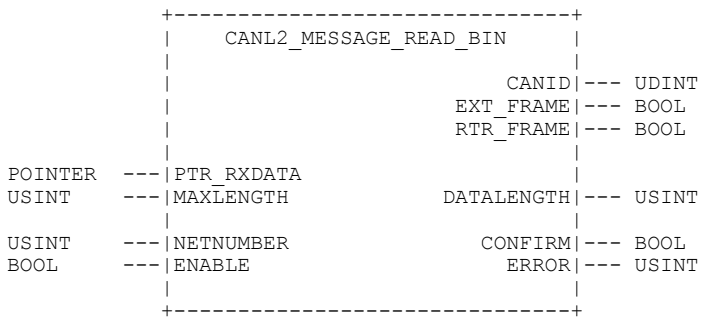
The function block CANL2_MESSAGE_READ8 serves for reading a received CAN-Message from the receive buffer of the CAN-Interface. Transfer of data occurs byte-by-byte at the outputs of the function block (alternatively: FB CANL2_MESSAGE_READ_BIN, see section 6.2.10). Only CAN-messages can be received for which appropriate CAN-objects have been specified previously by means of the function blocks CANL2_DEFINE_CANID or CANL2_DEFINE_CANID_RANGE respectively.

If the Output CONFIRM is set to TRUE during the return of the function block, the elements DATA0 to DATA7 contain the single bytes of the message. The output DATALENGTH shows the number of valid data bytes (from DATA0). Though the output CONFIRM is set to FALSE, the receive buffer of the CAN-Interface does not contain a message. With the help of CONFIRM it can be distinguished whether a valid message with a length of 0 bytes or no message has been received. If there is no message available, it is indicated through the error code NO_MESSAGE at the output ERROR (see Table 19, section 6.1.3).

6.2.10 Function Block CANL2_MESSAGE_READ_BIN

FB for reading a received CAN-message (transfer of data within the object addressed by pointer)

Prototype of the function block



Definition of Operands

PTR_RXDATA	Address of an object the data of the received CAN-message are stored in
MAX_LENGTH	Limitation of the number of stored bytes; at 0, the size of the object addressed via PTR_RXDATA is determined and as a limitation, the number of stored bytes is used (the number of stored bytes does not exceed the number of bytes the object is able to store).
DATALENGTH	Number of stored bytes
CANID	CAN-Identifier of the received CAN-Message
EXT_FRAME	TRUE = the CAN-Identifier marks an Extended-Frame (29 Bit) FALSE = the CAN-Identifier marks a Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Object has been passed as RTR-Frame FALSE = CAN-Object has not been passed as RTR-Frame
NETNUMBER	Network number
ERROR	Errorcode according to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

Description

The function block CANL2_MESSAGE_READ_BIN serves for reading a received CAN-message from the receive buffer of the CAN-interface. The data of the received message are stored within the object addressed via PTR_RXDATA (alternatively: FB CANL2_MESSAGE_READ8, see section 6.2.9). Only CAN-messages can be received, for which appropriate CAN-objects have been specified by means of the function blocks CANL2_DEFINE_CANID or CANL2_DEFINE_CANID_RANGE respectively.

By means of input MAX_LENGTH, the number of stored bytes can be limited. If MAX_LENGTH is set to 0, the size of the addressed object via PTR_RXDATA is determined internally and the number of bytes to be stored is used. The number of stored bytes does not exceed the number of bytes the object is able to store. Therewith, an overwriting of memory areas of other objects or variables is prevented.

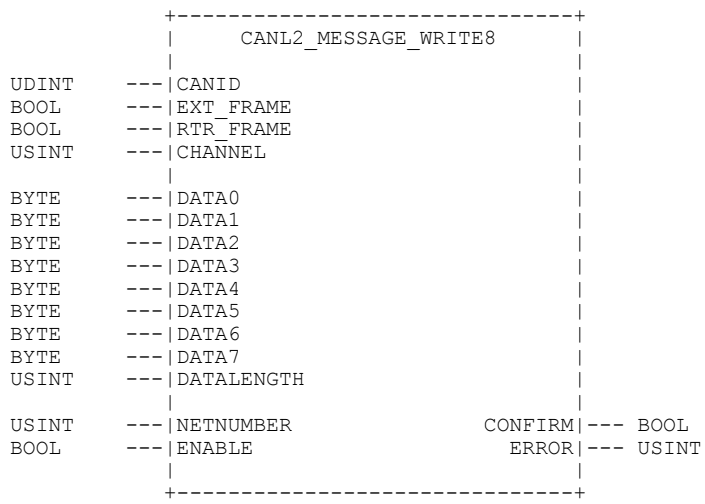
If the output CONFIRM is set to TRUE during the return of the function block, the object addressed via PTR_RXDATA contains the data of the received message. The output DATALENGTH specifies the number of valid data bytes. Is the output CONFIRM set to FALSE, however, the receive buffer does

not contain a message. With the help of CONFIRM it can be distinguished, whether a valid message with a length of 0 bytes or no message has been received. If there is no message available, it is indicated through the error code NO_MESSAGE at the output ERROR (see Table 19, section 6.1.3).

6.2.11 Function Block CANL2_MESSAGE_WRITE8

FB for sending a CAN-message (transfer of data on FB-inputs).

Prototype of the function block



Definition of Operands

CANID	CAN-Identifier of the CAN-message to be sent
EXT_FRAME	TRUE = the CAN-Identifier marks an Extended-Frame (29 Bit) FALSE = the CAN-Identifier marks a Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Object is passed as RTR-Frame FALSE = CAN-Object is not passed as RTR-Frame
CHANNEL	If the CAN-object has been defined by means of function block CANL2_DEFINE_CANID, then the channel number returned by this FB is to be passed for the CAN-object. If the CAN-object has been defined by means of function block CANL2_DEFINE_CANID_RANGE, then 0 has to be passed (Note: an object defined through CANL2_DEFINE_CANID_RANGE cannot be used for RTR-messages).
DATA0 - DATA7	Data bytes of the CAN-message to be sent
DATALENGTH	Length of the CAN-message to be sent
NETNUMBER	Network number
ERROR	Errorcode according to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

Description

The function block CANL2_MESSAGE_WRITE8 serves for sending a CAN-Message. The transfer of data occurs byte-by-byte at the inputs of the function block (alternatively: FB CANL2_MESSAGE_WRITE_BIN, see section 6.2.12). Only CAN-messages can be sent for which appropriate CAN-objects have been designed by means of function blocks CANL2_DEFINE_CANID or CANL2_DEFINE_CANID_RANGE.

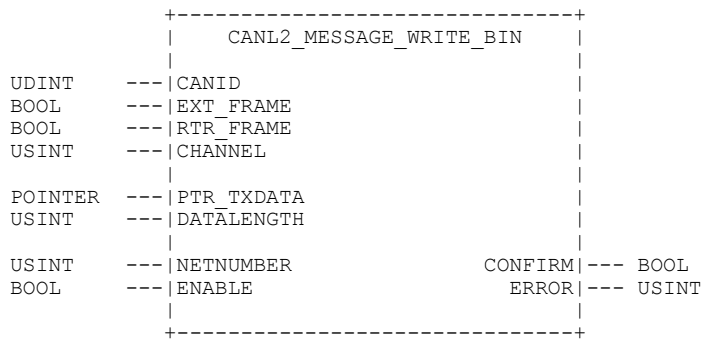
At the elements DATA0 to DATA7, the single bytes of the message to be sent are to pass. The input DATALENGTH thereby specifies the number of valid data bytes (from DATA0).

When calling function block CANL2_MESSAGE_WRITE8, the message to be sent is stored in the send buffer of the CAN-Interface. In case no error occurs (message could be stored properly in the send buffer), the module returns with the output CONFIRM, which is set to TRUE. However, there is no feedback signal to the PLC program, whether the message could be sent successfully.

6.2.12 Function Block CANL2_MESSAGE_WRITE_BIN

FB for sending a CAN-Message (transfer of data within the object addressed by pointer)

Prototype of the function block



Definition of Operands

CANID	CAN-Identifier of the CAN-message to be sent
EXT_FRAME	TRUE = the CAN-Identifier marks an Extended-Frame (29 Bit) FALSE = the CAN-Identifier marks a Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Object is passed as RTR-Frame FALSE = CAN-Object is not passed as RTR-Frame
CHANNEL	If the CAN-Object has been defined by means of function block CANL2_DEFINE_CANID, then the channel number returned from this function block for the CAN-object has to be passed. If the CAN-object has been defined by means of function block CANL2_DEFINE_CANID_RANGE, then 0 has to be passed (Note: a CAN-object defined through CANL2_DEFINE_CANID_RANGE cannot be used for RTR-messages).
PTR_TXDATA	Address of an object containing the data to be sent with the CAN-message
DATALENGTH	Number of bytes to be sent through the CAN-message; at 0, the size of the object addressed via PTR_TXDATA is determined and the number of bytes that have to be sent is used.
NETNUMBER	Network number

ERROR	Errorcode according to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

Description

The function block CANL2_MESSAGE_WRITE_BIN serves for sending a CAN-message. The data of the message to be sent are expected in the object addressed via PTR_TXDATA (alternatively: FB CANL2_MESSAGE_WRITE8, see section 6.2.11). Only CAN-messages can be sent, for which appropriate CAN-objects have been designed by means of function blocks CANL2_DEFINE_CANID or CANL2_DEFINE_CANID_RANGE respectively.

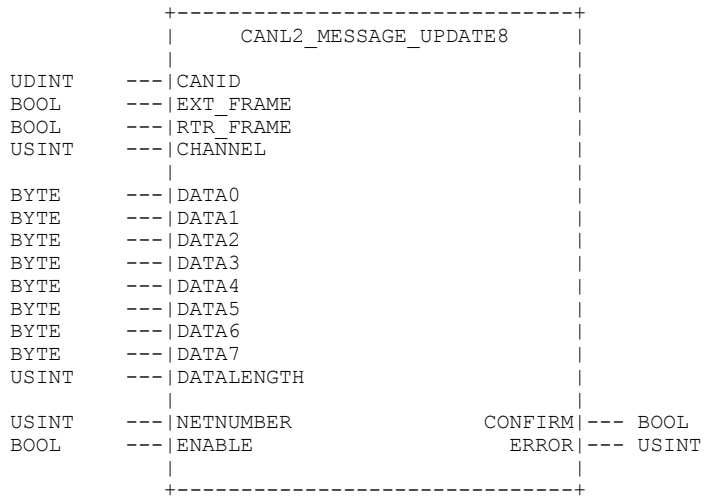
The data of the message to be sent are taken from the object addressed via PTR_TXDATA By means of the input DATALENGTH, the number of bytes to be sent can be limited. If DATALENGTH is set to 0, the size of the object addressed via PTR_TXDATA is determined internally and used as the number of bytes to be sent.

When calling function block CANL2_MESSAGE_WRITE_BIN, the message to be sent is stored in the send buffer of the CAN-interface. If no error occurs here (message could be stored properly in the send buffer), the module returns with the output CONFIRM, which is set to TRUE. However, there is no feedback signal to the PLC program, whether the message could be sent successfully.

6.2.13 Function Block CANL2_MESSAGE_UPDATE8

FB to update data of a RTR-Message (transfer of data on FB-inputs)

Prototype of the function block



Definition of Operands

CANID	CAN-Identifier of the CAN-Message to be updated
EXT_FRAME	TRUE = the CAN-Identifier marks an Extended-Frame (29 Bit) FALSE = the CAN-Identifier marks a Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Object is passed as RTR-Frame FALSE = CAN-Object is not passed as RTR-Frame

CHANNEL	Channel number that has been returned during the definition of the CAN-Object from function block CANL2_DEFINE_CANID (Note: CAN-objects defined with CANL2_DEFINE_CANID_RANGE cannot be used for RTR-Messages)
DATA0 - DATA7	Data bytes of the CAN-message to be updated
DATALENGTH	Length of the CAN-message to be updated
NETNUMBER	Network number
ERROR	Errorcode relates to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

Description

The function block CANL2_MESSAGE_UPDATE8 serves for updating data of a RTR-message. The transfer of data occurs byte-by-byte at the inputs of the function block (alternatively: FB CANL2_MESSAGE_UPDATE_BIN, see section 6.2.14). Only CAN-objects can be updated that have been designed previously by means of the function block CANL2_DEFINE_CANID.

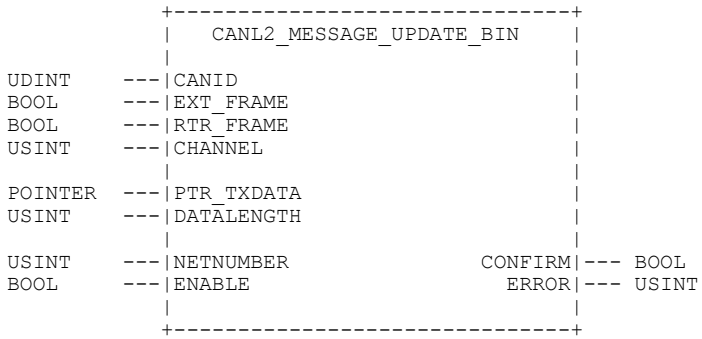
On the elements DATA0 to DATA7, the single bytes have to be passed for the message to be updated. The input DATALENGTH thereby specifies the number of valid data bytes (from DATA0).

When calling function block CANL2_MESSAGE_UPDATE8, only the data of messages in the send buffer of the CAN-controller are updated. To transfer the message on the CAN-bus, this has to be requested explicitely by another node through RTR-Frame.

6.2.14 Function Block CANL2_MESSAGE_UPDATE_BIN

FB for the update of data of an RTR-Message (transfer of data within the object addressed through pointer)

Prototype of the function block



Definition of Operands

CANID	CAN-Identifier of the CAN-message to be updated
EXT_FRAME	TRUE = the CAN-Identifier marks an Extended-Frame (29 Bit) FALSE = the CAN-Identifier marks a Standard-Frame (11 Bit)
RTR_FRAME	TRUE = CAN-Object is passes as RTR-Frame FALSE = CAN-Object is not passed as RTR-Frame

CHANNEL	Channel number, which has been returned from function block CANL2_DEFINE_CANID during the definition of the CAN-Objects (Note: CAN-objects defined through CANL2_DEFINE_CANID_RANGE cannot be used for RTR-messages)
PTR_TXDATA	Address of an object that contains the data of the CAN-message to be updated
DATALENGTH	Number of bytes in the CAN-message to be updated; at 0, the size of the object addressed via PTR_TXDATA is determined internally and used as number of bytes to be updated
NETNUMBER	Network number
ERROR	Errorcode according to data type "CANL2_ERROR" (see Table 19, section 6.1.3)
ENABLE	Input for enabling or disabling of the FB (see section 6.1.2)
CONFIRM	Output for message completed through the FB (see section 6.1.2)

Description

The function block CANL2_MESSAGE_UPDATE_BIN serves for updating data of an RTR-message. The data to be updated are expected in the object addressed via PTR_TXDATA (alternatively: FB CANL2_MESSAGE_UPDATE8, see section 6.2.13). Only CAN-objects can be updated that have been desgined previously with the help of function block CANL2_DEFINE_CANID.

The data of the message to be updated are taken from the object addressed via PTR_TXDATA. By means of input DATALENGTH, the number of bytes that have to be updated can be limited. If DATALENGTH is set to 0, the size of the object addressed via PTR_TXDATA is determined internally and the number of bytes to be updated is used.

When calling function block CANL2_MESSAGE_UPDATE_BIN, only the data of the message in the send buffer of the CAN-controller are updated. To transfer the message on the CAN-bus, this has to be requested explicitely by another node via RTR-Frame.

7 Index

Document:	**CAN / CANopen Extenstion for IEC 61131**
Document number:	**L-1008-07, May 2011**

Do you have any suggestions for improving this manual?

Have you discovered any errors in this manual? Page

Sent by:

Customer number:

Name:

Company:

Address:

Send to:

SYS TEC electronic GmbH
Am Windrad 2
D – 08468 Heinsdorfergrund
GERMANY
Fax: +49 (0) 37 65 / 38600-4100
Email: info@systec-electronic.com