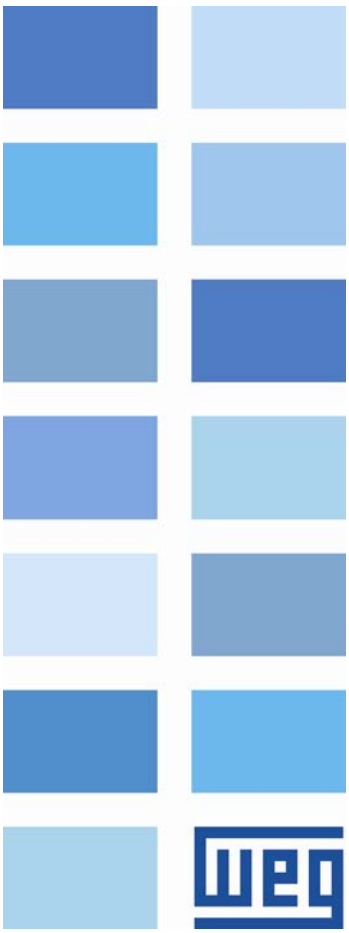
CANopen

PLC300

User's Manual



CANopen User's Manual

Series: PLC300

Language: English

Document Number: 10003124832 / 00

Publication Date: 10/2014

CONTENTS

ABOUT THE MANUAL

This manual provides the necessary information for the operation of the PLC300 frequency inverter using the CANopen protocol. This manual must be used together with the PLC300 user manual.

ABBREVIATIONS AND DEFINITIONS

CAN	Controller Area Network
CiA	CAN in Automation
COB	Communication Object
COB-ID	Communication Object Identifier
SDO	Service Data Object
PDO	Process Data Object
RPDO	Receive PDO
TPDO	Transmit PDO
NMT	Network Management Object
ro	Read only
rw	Read/write

NUMERICAL REPRESENTATION

Decimal numbers are represented by means of digits without suffix. Hexadecimal numbers are represented with the letter 'h' after the number.

DOCUMENTS

The CANopen protocol for the PLC300 was developed based on the following specifications and documents:

Document	Version	Source
CAN Specification	2.0	CiA
CiA DS 301 CANopen Application Layer and Communication Profile	4.02	CiA
CiA DRP 303-1 Cabling and Connector Pin Assignment	1.1.1	CiA
CiA DSP 306 Electronic Data Sheet Specification for CANopen	1.1	CiA
CiA DSP 402 Device Profile Drives and Motion Control	2.0	CiA

In order to obtain this documentation, the organization that maintains, publishes and updates the information regarding the CANopen network, CiA, must be consulted.

1 INTRODUCTION TO THE CANOPEN COMMUNICATION

In order to operate the equipment in a CANopen network, it is necessary to know the manner this communication is performed. Therefore, this section brings a general description of the CANopen protocol operation, containing the functions used by the PLC300. Refer to the protocol specification for a detailed description.

1.1 CAN

CANopen is a network based on CAN, i.e., it uses CAN telegrams for exchanging data in the network.

The CAN protocol is a serial communication protocol that describes the services of layer 2 of the ISO/OSI model (data link layer)[1]. This layer defines the different types of telegrams (frames), the error detection method, the validation and arbitration of messages.

1.1.1 Data Frame

CAN network data is transmitted by means of a data frame. This frame type is composed mainly by an 11 bit[2] identifier (arbitration field), and by a data field that may contain up to 8 data bytes.

Identifier	8 data bytes							
11 bits	byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7

1.1.2 Remote Frame

Besides the data frame, there is also the remote frame (RTR frame). This type of frame does not have a data field, but only the identifier. It works as a request, so that another network device transmits the desired data frame.

1.1.3 Access to the Network

Any device in a CAN network can make an attempt to transmit a frame to the network in a certain moment. If two devices try to access the network simultaneously, the one that sends the message with the highest priority will be able to transmit. The message priority is defined by the CAN frame identifier, the smaller the value of this identifier, the higher the message priority. The telegram with the identifier 0 (zero) is the one with the highest priority.

1.1.4 Error Control

The CAN specification defines several error control mechanisms, which makes the network very reliable and with a very low undetected transmission error rate. Every network device must be able to identify the occurrence of these errors, and to inform the other elements that an error was detected.

A CAN network device has internal counters that are incremented every time a transmission or reception error is detected, and are decremented when a telegram is successfully transmitted or received. If a considerable amount of errors occurs, the device can be led to the following states:

- *Error Active:* the internal error counters are at a low level and the device operates normally in the CAN network. You can send and receive telegrams and act in the CAN network if it detects any error in the transmission of telegrams.
- *Warning:* when the counter exceeds a defined limit, the device enters the *warning* state, meaning the occurrence of a high error rate.
- *Error Passive:* when this value exceeds a higher limit, the device enters the *error passive* state, and it stops acting in the network when detecting that another device sent a telegram with an error.
- *Bus Off:* finally, we have the *bus off* state, in which the device will not send or receive telegrams any more. The device operates as if disconnected from the network.

[1] In the CAN protocol specification, the ISO11898 standard is referenced as the definition of the layer 1 of this model (physical layer).
[2] The CAN 2.0 specification defines two data frame types, standard (11 bit) and extended (29 bit). For this implementation, only the standard frames are accepted.

1.1.5 CAN and CANopen

Only the definition of how to detect errors, create and transmit a frame, are not enough to define a meaning for the data transmitted via the network. It is necessary to have a specification that indicates how the identifier and the data must be assembled and how the information must be exchanged. Thus, the network elements can interpret the transmitted data correctly. In that sense, the CANopen specification defines exactly how to exchange data among the devices and how every one must interpret these data.

There are several other protocols based on CAN, as DeviceNet, CANopen, J1939, etc., which use CAN frames for the communication. However, those protocols cannot be used together in the same network.

1.2 NETWORK CHARACTERISTICS

Because of using a CAN bus as telegram transmission means, all the CANopen network devices have the same right to access the network, where the identifier priority is responsible for solving conflict problems when simultaneous access occurs. This brings the benefit of making direct communication between slaves of the network possible, besides the fact that data can be made available in a more optimized manner without the need of a master that controls all the communication performing cyclic access to all the network devices for data updating.

Another important characteristic is the use of the producer/consumer model for data transmission. This means that a message that transits in the network does not have a fixed network address as a destination. This message has an identifier that indicates what data it is transporting. Any element of the network that needs to use that information for its operation logic will be able to consume it, therefore, one message can be used by several network elements at the same time.

1.3 PHYSICAL LAYER

The physical medium for signal transmission in a CANopen network is specified by the ISO 11898 standard. It defines as transmission bus a pair of twisted wires with differential electrical signal.

1.4 ADDRESS IN THE CANOPEN NETWORK

Every CANopen network must have a master responsible for network management services, and it can also have a set of up to 127 slaves. Each network device can also be called node. Each slave is identified in a CANopen network by its address or Node-ID, which must be unique for each slave and may range from 1 to 127.

The address of programmable controller PLC300 is programmed by the menu Setup.

1.5 ACCESS TO THE DATA

Each slave of the CANopen network has a list called object dictionary that contains all the data accessible via network. Each object of this list is identified with an index, which is used during the equipment configuration as well as during message exchanges. This index is used to identify the object being transmitted.

1.6 DATA TRANSMISSION

The transmission of numerical data via CANopen telegrams is done using a hexadecimal representation of the number, and sending the least significant data byte first.

E.g: The transmission of a 32 bit integer with sign (12345678h = 305419896 decimal), plus a 16 bit integer with sign (FF00h = -256 decimal), in a CAN frame.

Identifier	6 data bytes					
11 bits	32 bit integer				16 bit integer	
	byte 0	byte 1	byte 2	byte 3	byte 4	byte 5
	78h	56h	34h	12h	00h	FFh

1.7 COMMUNICATION OBJECTS - COB

There is a specific set of objects that are responsible for the communication among the network devices. Those objects are divided according to the type of data and the way they are sent or received by a device. The PLC300 supports the following communication objects (COB):

Table 1.1: *Types of Communication Objects (COB)*

Type of object	Description
Service Data Object (SDO)	SDO are objects responsible for the direct access to the object dictionary of a device. By means of messages using SDO, it is possible to indicate explicitly (by the object index) what data is being handled. There are two SDO types: Client SDO, responsible for doing a read or write request to a network device, and the Server SDO, responsible for taking care of that request. Since SDO are usually used for the configuration of a network node, they have less priority than other types of message.
Process Data Object (PDO)	PDO are used for accessing equipment data without the need of indicating explicitly which dictionary object is being accessed. Therefore, it is necessary to configure previously which data the PDO will be transmitting (data mapping). There are also two types of PDO: Receive PDO and Transmit PDO. They are usually utilized for transmission and reception of data used in the device operation, and for that reason they have higher priority than the SDO.
Emergency Object (EMCY)	This object is responsible for sending messages to indicate the occurrence of errors in the device. When an error occurs in a specific device (EMCY producer), it can send a message to the network. In the case that any network device be monitoring that message (EMCY consumer), it can be programmed so that an action be taken (disabling the other devices, error reset, etc.).
Synchronization Object (SYNC)	In the CANopen network, it is possible to program a device (SYNC producer) to send periodically a synchronization message for all the network devices. Those devices (SYNC consumers) will then be able, for instance, to send a certain datum that needs to be made available periodically.
Network Management (NMT)	Every CANopen network needs a master that controls the other devices (slaves) in the network. This master will be responsible for a set of services that control the slave communications and their state in the CANopen network. The slaves are responsible for receiving the commands sent by the master and for executing the requested actions. The protocol describes two types of service that the master can use: device control service, with which the master controls the state of each network slave, and error control service (Node Guarding), with which the slave sends periodic messages to the master informing that the connection is active.

All the communication of the inverter with the network is performed using those objects, and the data that can be accessed are the existent in the device object dictionary.

1.8 COB-ID

A telegram of the CANopen network is always transmitted by a communication object (COB). Every COB has an identifier that indicates the type of data that is being transported. This identifier, called COB-ID has an 11 bit size, and it is transmitted in the identifier field of a CAN telegram. It can be subdivided in two parts:

Function Code				Address						
bit 10	bit 9	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

- Function Code: indicates the type of object that is being transmitted.
- Node Address: indicates with which network device the telegram is linked.

A table with the standard values for the different communication objects available in the PLC300 is presented next. Notice that the standard value of the object depends on the slave address, with the exception of the COB-ID for NMT and SYNC, which are common for all the network elements. Those values can also be changed during the device configuration stage.

Table 1.2: *COB-ID for the different objects*

COB	Function code (bits 10 – 7)	Resultant COB-ID (function + address)
NMT	0000	0
SYNC	0001	128 (80h)
EMCY	0001	129 – 255 (81h – FFh)
PDO1 (tx)	0011	385 – 511 (181h – 1FFh)
PDO1 (rx)	0100	513 – 639 (201h – 27Fh)
PDO2 (tx)	0101	641 – 767 (281h – 2FFh)
PDO2 (rx)	0110	769 – 895 (301h – 37Fh)
PDO3 (tx)	0111	897 – 1023 (381h – 3FFh)
PDO3 (rx)	1000	1025 – 1151 (401h – 47Fh)
PDO4 (tx)	1001	1153 – 1279 (481h – 4FFh)
PDO4 (rx)	1010	1281 – 1407 (501h – 57Fh)
SDO (tx)	1011	1409 – 1535 (581h – 5FFh)
SDO (rx)	1100	1537 – 1663 (601h – 67Fh)
Node Guarding/Heartbeat	1110	1793 – 1919 (701h – 77Fh)

1.9 EDS FILE

Each device in a CANopen network has an EDS configuration file that contains information about the operation of the device in the CANopen network, as well as the description of all the communication objects available. In general, this file is used by a master or by the configuration software for programming of devices present in the CANopen Network.

The EDS configuration file for the PLC300 is supplied together with the product, and it can also be obtained from the website http://www.weg.net. It is necessary to observe the inverter software version, in order to use an EDS file that be compatible with that version.

2 CAN COMMUNICATION INTERFACE

The programmable controller PLC300 has a CAN interface in the standard product. It is possible to use it for communication on the CANopen protocol as network master or slave. Features of the interface are described below.

2.1 CAN INTERFACE CHARACTERÍSTICS

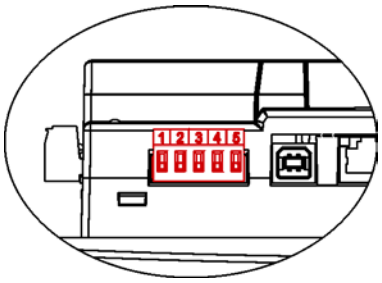


Figure 2.1: *CAN connector detail on the bottom of the product*

- The interface is electrically isolated and with differential signal, which grants more robustness against electromagnetic interference.
- External 24V supply.
- It allows the connection of up to 64 devices to the same segment. More devices can be connected by using repeaters.
- A maximum bus length of 1000 meters.

2.2 CONNECTOR PINOUT

The CAN communication module presents a 5-wire plug-in connector (XC6) with the following pinout:

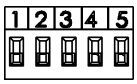


Table 2.1: *CAN interface XC5 connector pinout*

Pin	Name	Function
1	V-	Power supply negative pole
2	CAN_L	CAN_L communication signal
3	Shield	Cable shield
4	CAN_H	CAN_H communication signal
5	V+	Power supply positive pole

2.3 POWER SUPPLY

The CAN interface needs an external power supply between the pins 1 and 5 of the network connector. The individual consumption and input voltage data are presented in the next table.

Table 2.2: *CAN interface supply characteristics*

Supply Voltage (V_{DC})		
Minimum	Maximum	Recommended
11	30	24
Current (mA)		
Typical		Maximum
30		50

2.4 INDICATIONS

Besides the system markers, which provide information about the interface, the device has a bicolor LED – green and red – in the front of the product, to indicate the CAN interface status.

Figure 2.2: *Indication LED of CAN interface*

During equipment startup, both LEDs are lit to test for a period of approximately 500 ms alternately. After this period, for the CANopen protocol, they will indicate as shown below.

2.4.1 Indication Patterns

In addition to the ON and OFF states, the following behaviors may also be displayed:

- Blinking: LED is a period of 200 ms on, followed by a period of 200 ms off.
- One flash: the LED is a period of 200 ms on, followed by a period of 1 second off.
- Two flashes: the LED lights up twice for a period of 200 ms (with a period of 200 ms off between these indications), followed by a period of 1 second off.

2.4.2 Error LED (red)

The red LED indicates errors of the physical layer of the CAN bus, and CANopen communication errors.

Table 2.3: *Indications for Error LED (red)*

Indication	State	Description
Off	No error / No power	The device is operating normally, shut down or unpowered CAN interface.
One Flash	Warning state	The internal error counters of the CAN controller reached the warning state due to errors of CAN communication. This statement is also valid if the equipment is in error passive state.
Two Flashes	Error control services – Node Guarding or Heartbeat	After the master initializes the Node Guarding or Heartbeat services, an exchange timeout occurred between the master and slave, causing this error.
On	Bus off	The CAN controller has reached the bus off state. Communication is disabled.

2.4.3 Run LED (green)

The green LED indicates the status of the CANopen slave.

Tabela 2.4: *Indication for Run LED (green)*

Indication	State	Description
One flash	STOPPED	The device is in the stopped state.
Blinking	PRE-OPERATIONAL	The device is in the pre-operational state.
On	OPERATIONAL	The device is in operational state.

NOTE!
If the state of the CAN interface and CANopen communication is such that both LEDs should make simultaneous indications, the red LED will take precedence over the green LED, the latter remaining off.

3 CANOPEN NETWORK INSTALLATION

The CANopen network, such as several industrial communication networks, for being many times applied in aggressive environments with high exposure to electromagnetic interference, requires that certain precautions be taken in order to guarantee a low communication error rate during its operation. Recommendations to perform the connection of the product in this network are presented next.

3.1 BAUD RATE

Equipments with CANopen interface generally allow the configuration of the desired baud rate, ranging from 10Kbit/s to 1Mbit/s. The *baud rate* that can be used by equipment depends on the length of the cable used in the installation. The next table shows the baud rates and the maximum cable length that can be used in the installation, according to the CiA recommendation[3].

Table 3.1: *Supported baud rates and installation size*

Baud Rate	Cable Length
1 Mbit/s	25 m
800 Kbit/s	50 m
500 Kbit/s	100 m
250 Kbit/s	250 m
125 Kbit/s	500 m
100 Kbit/s	600 m
50 Kbit/s	1000 m
20 Kbit/s	1000 m
10 Kbit/s	1000 m

All network equipment must be programmed to use the same communication baud rate. At the PLC300 programmable controller the baud rate configuration is done through the Setup menu.

3.2 ADDRESS IN THE CANOPEN NETWORK

Each CANopen network device must have an address or Node ID, and may range from 1 to 127. This address must be unique for each equipment. For PLC300 programmable controller the address configuration is done through the Setup menu.

3.3 TERMINATION RESISTOR

The CAN bus line must be terminated with resistors to avoid line reflection, which can impair the signal and cause communication errors. The extremes of the CAN bus must have a termination resistor with a 121Ω / 0.25W value, connecting the CAN_H and CAN_L signals.

3.4 CABLE

The connection of CAN_L and CAN_H signals must done with shielded twisted pair cable. The following table shows the recommended characteristics for the cable.

Table 3.2: *CANopen cable characteristics*

Cable length (m)	Resistance per meter (mOhm/m)	Conductor cross section (mm^2)
0 ... 40	70	0.25 ... 0.34
40 ... 300	<60	0.34 ... 0.60
300 ... 600	<40	0.50 ... 0.60
600 ... 1000	<26	0.75 ... 0.80

It is necessary to use a twisted pair cable to provide additional 24Vdc power supply to equipments that need this signal. It is recommended to use a certified DeviceNet cable.

[3] Different products may have different maximum allowed cable length for installation.

3.5 CONNECTION IN THE NETWORK

In order to interconnect the several network nodes, it is recommended to connect the equipment directly to the main line without using derivations. During the cable installation the passage near to power cables must be avoided, because, due to electromagnetic interference, this makes the occurrence of transmission errors possible. In order to avoid problems with current circulation caused by difference of potential among ground connections, it is necessary that all the devices be connected to the same ground point.

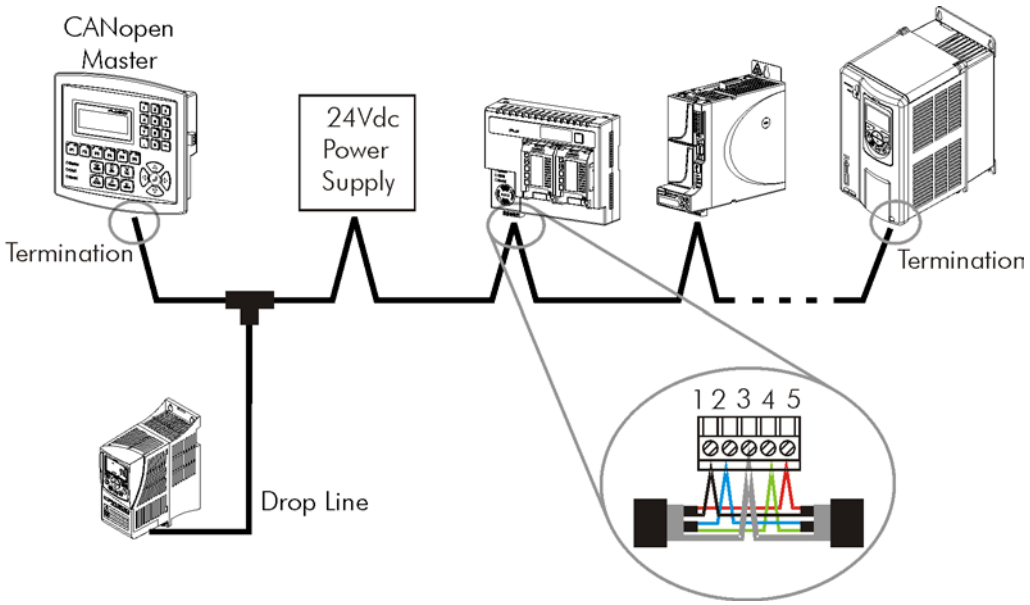


Figure 3.1: *CANopen network installation example*

To avoid voltage difference problems between the power supplies of the network devices, it is recommended that the network is fed by only one power supply and the signal is provided to all devices through the cable. If it is required more than one power supply, these should be referenced to the same point.

The maximum number of devices connected to a single segment of the network is limited to 64. Repeaters can be used for connecting a bigger number of devices.

4 CAN INTERFACE CONFIGURATION

To perform configuration of the CAN interface, the PLC300 Setup has the following menus:

BAUD RATE

Range:	0 = 1 Mbit/s 1 = Reserved 2 = 500 Kbit/s 3 = 250 Kbit/s 4 = 125 Kbit/s 5 = 100 Kbit/s 6 = 50 Kbit/s 7 = 20 Kbit/s	**Default:** 0

Description:
It allows programming the desired baud rate for the CAN interface, in bits per second. This rate must be the same for all the devices connected to the network.

If this parameter is changed, the change takes effect only if the CAN interface is not powered or after the equipment is switched off and on again.

ADDRESS

Range:	1 a 127	**Default:** 1

Description:
It allows programming the address used for the CAN communication. It is necessary that each element of the network has an address different from the others.

If this parameter is changed, the change takes effect only if the CAN interface is not powered or after the equipment is switched off and on again.

5 OBJECT DICTIONARY

The object dictionary is a list containing several equipment data which can be accessed via CANopen network. An object of this list is identified by means of a 16-bit index, and it is based in that list that all the data exchange between devices is performed.

The CiA DS 301 document defines a set of minimum objects that every CANopen network slave must have. The objects available in that list are grouped according to the type of function they execute. The objects are arranged in the dictionary in the following manner:

Table 5.1: *Object dictionary groupings*

Index	Objects	Description
0001h – 025Fh	Data type definition	Used as reference for the data type supported by the system.
1000h – 1FFFh	Communication objects	They are objects common to all the CANopen devices. They contain general information about the equipment and also data for the communication configuration.
2000h – 5FFFh	Manufacturer specific objects	In this range, each CANopen equipment manufacturer is free to define which data those objects will represent.
6000h – 9FFFh	Standardized device objects	This range is reserved to objects that describe the behavior of similar equipment, regardless of the manufacturer.

The other indexes that are not referred in this list are reserved for future use.

5.1 DICTIONARY STRUCTURE

The general structure of the dictionary has the following format:

Index	Object	Name	Type	Access

- Index: indicates directly the object index in the dictionary.
- Object: describes which information the index stores (simple variable, array, record, etc.).
- Name: contains the name of the object in order to facilitate its identification.
- Type: indicates directly the stored data type. For simple variables, this type may be an integer, a float, etc. For arrays, it indicates the type of data contained in the array. For records, it indicates the record format according to the types described in the first part of the object dictionary (indexes 0001h – 0360h).
- Access: informs if the object in question is accessible only for reading (ro), for reading and writing (rw), or if it is a constant (const).

For objects of the array or record type, a sub-index that is not described in the dictionary structure is also necessary.

5.2 DATA TYPE

The first part of the object dictionary (index 0001h – 025Fh) describes the data types that can be accessed at a CANopen network device. They can be basic types, as integers and floats, or compound types formed by a set of entries, as records and arrays.

5.3 COMMUNICATION PROFILE – COMMUNICATION OBJECTS

The indexes from 1000h to 1FFFh in the object dictionary correspond to the part responsible for the CANopen network communication configuration. Those objects are common to all the devices, however only a few are obligatory. A list with the objects of this range that are supported by the programmable controller PLC300, working in slave mode.is presented next.

Table 5.2: *Object list – Communication Profile*

Índice	Objeto	Nome	Tipo	Acesso
1000h	VAR	device type	UNSIGNED32	ro
1001h	VAR	error register	UNSIGNED8	ro
1005h	VAR	COB-ID SYNC	UNSIGNED32	rw
100Ch	VAR	guard time	UNSIGNED16	rw
100Dh	VAR	life time factor	UNSIGNED8	rw
1016h	ARRAY	Consumer heartbeat time	UNSIGNED32	rw
1017h	VAR	Producer heartbeat time	UNSIGNED16	rw
1018h	RECORD	Identity Object	Identity	ro
Server SDO Parameter				
1200h	RECORD	1st Server SDO parameter	SDO Parameter	ro
Receive PDO Communication Parameter				
1400h	RECORD	1st receive PDO Parameter	PDO CommPar	rw
1401h	RECORD	2nd receive PDO Parameter	PDO CommPar	rw
...				
1407h	RECORD	8th receive PDO Parameter	PDO CommPar	rw
Receive PDO Mapping Parameter				
1600h	RECORD	1st receive PDO mapping	PDO Mapping	rw
1601h	RECORD	2nd receive PDO mapping	PDO Mapping	rw
...				
1607h	RECORD	8th receive PDO mapping	PDO Mapping	rw
Transmit PDO Communication Parameter				
1800h	RECORD	1st transmit PDO Parameter	PDO CommPar	rw
1801h	RECORD	2nd transmit PDO Parameter	PDO CommPar	rw
...				
1807h	RECORD	8th transmit PDO Parameter	PDO CommPar	rw
Transmit PDO Mapping Parameter				
1A00h	RECORD	1st transmit PDO mapping	PDO Mapping	rw
1A01h	RECORD	2nd transmit PDO mapping	PDO Mapping	rw
...				
1A07h	RECORD	8th transmit PDO mapping	PDO Mapping	rw

These objects can only be read and written via the CANopen network, it is not available via the keypad (HMI) or other network interface. The network master, in general, is the equipment responsible for setting up the equipment before starting the operation. The EDS configuration file brings the list of all supported communication objects.

Refer to item 6 for more details on the available objects in this range of the objects dictionary.

5.4 MANUFACTURER SPECIFIC OBJECTS

For indexes from 2000h to 5FFFh, each manufacture is free to define which objects will be present, and also the type and function of each one. For PLC300, this object range has the network markers. The product can communicate these markers through the CANopen interface and use them in the controller programming software for designing the operation logic of the equipment. The next table illustrates how the markers are distributed in the object dictionary.

Table 5.1: *Object list – Manufacturer Specific*

Index	Object	Name	Type	Access
Network Input Data – Byte Access				
3000h	VAR	Network Input Byte 2000 – %IB2000	UNSIGNED8	rw
3001h	VAR	Network Input Byte 2001 – %IB2001	UNSIGNED8	rw
3002h	VAR	Network Input Byte 2002 – %IB2002	UNSIGNED8	rw
...	...	...	...	...
31FFh	VAR	Network Input Byte 2511 – %IB2511	UNSIGNED8	rw
Network Input Data – Word Access				
3400h	VAR	Network Input Word 2000 – %IW2000	UNSIGNED16	rw
3402h	VAR	Network Input Word 2002 – %IW2002	UNSIGNED16	rw
3404h	VAR	Network Input Word 2004 – %IW2004	UNSIGNED16	rw
...	...	...	...	...
35FEh	VAR	Network Input Word 2510 – %IW2510	UNSIGNED16	rw
Network Input Data – Double Word Access				
3800h	VAR	Network Input Double Word 2000 – %ID2000	UNSIGNED32	rw
3804h	VAR	Network Input Double Word 2004 – %ID2004	UNSIGNED32	rw
3808h	VAR	Network Input Double Word 2008 – %ID2008	UNSIGNED32	rw
...	...	...	...	...
39FCh	VAR	Network Input Double Word 2508 – %ID2508	UNSIGNED32	rw
Network Output Data – Byte Access				
4000h	VAR	Network Output Byte 2000 – %QB2000	UNSIGNED8	rw
4001h	VAR	Network Output Byte 2001 – %QB2001	UNSIGNED8	rw
4002h	VAR	Network Output Byte 2002 – %QB2002	UNSIGNED8	rw
...	...	...	...	...
41FFh	VAR	Network Output Byte 2511 – %QB2511	UNSIGNED8	rw
Network Output Data – Word Access				
4400h	VAR	Network Output Word 2000 – %QW2000	UNSIGNED16	rw
4402h	VAR	Network Output Word 2002 – %QW2002	UNSIGNED16	rw
4404h	VAR	Network Output Word 2004 – %QW2004	UNSIGNED16	rw
...	...	...	...	...
45FEh	VAR	Network Output Word 2510 – % QW2510	UNSIGNED16	rw
Network Output Data – Double Word Access				
4800h	VAR	Network Output Double Word 2000 – %QD2000	UNSIGNED32	rw
4804h	VAR	Network Output Double Word 2004 – %QD2004	UNSIGNED32	rw
4808h	VAR	Network Output Double Word 2008 – %QD2008	UNSIGNED32	rw
...	...	...	...	...
49FCh	VAR	Network Output Double Word 2508 – %QD2508	UNSIGNED32	rw

NOTE!

- Input Markers of Byte, Word and Double Word share the same internal memory area in the product. Thus, for example, the markers %IB2000 and %IB2001 occupy the same memory area that the marker %IW2000. Different objects exist only to provide objects of different sizes for mapping data via CANopen. The same goes for the Output area.
- The data types used in these objects are defined as unsigned integer of 8, 16 or 32 bit. This type is only to define the size of the data in CANopen communication. The actual markers type, however, depends on the declared type in the controller programming software. The %QD2000 marker, for example, may represent a given type float, depending on what was stated in the programming software.
- Network Input Markers can be mapped in the receive PDOs, while Network Output Markers can be mapped to transmit PDOs.

6 COMMUNICATION OBJECTS DESCRIPTION

This item describes in detail each of the communication objects available for the programmable controller PLC300 working in slave mode. It is necessary to know how to operate these objects to be able to use the available functions for the inverter communication.

NOTE!
The programmable controller PLC300 can operate as master or slave of the CANopen network. The objects below describe the operation of the equipment as slave of the CANopen network. For a description of the characteristics of the product operating as CANopen network master, refer to the item 7 together with the WSCAN CANopen network configuration software.

6.1 IDENTIFICATION OBJECTS

There is a set of objects in the dictionary which are used for equipment identification; however, they do not have influence on their behavior in the CANopen network.

6.1.1 Object 1000h – Device Type

This object gives a 32-bit code that describes the type of object and its functionality.

Index	1000h
Name	Device type
Object	VAR
Type	UNSIGNED32

Access	ro
PDO Mapping	No
Range	UNSIGNED32
Default value	0000.0000h

This code can be divided into two parts: 16 low-order bits describing the type of profile that the device uses, and 16 high-order bits indicating a specific function according to the specified profile.

6.1.2 Object 1001h – Error Register

This object indicates whether or not an error in the device occurred. The type of error registered for the PLC300 follows what is described in the table 6.1.

Index	1001h
Name	Error register
Object	VAR
Type	UNSIGNED8

Access	ro
PDO Mapping	Yes
Range	UNSIGNED8
Default value	0

Table 6.1: *Structure of the object Error Register*

Bit	Meaning
0	Generic error
1	Current
2	Voltage
3	Temperature
4	Communication
5	Reserved (always 0)
6	Reserved (always 0)
7	Specific of the manufacturer

If the device presents any error, the equivalent bit must be activated. The first bit (generic error) must be activated with any error condition.

6.1.3 Object 1018h – Identity Object

It brings general information about the device.

Index	1018h
Name	Identity object
Object	Record
Type	Identity

Sub index	0
Description	Number of the last sub-index
Access	RO
PDO Mapping	No
Range	UNSIGNED8
Default value	4

Sub index	1
Description	Vendor ID
Access	RO
PDO Mapping	No
Range	UNSIGNED32
Default value	0000.0123h

Sub index	2
Description	Product code
Access	RO
PDO Mapping	No
Range	UNSIGNED32
Default value	0000.0220h

Sub index	3
Description	Revision number
Access	RO
PDO Mapping	No
Range	UNSIGNED32
Default value	According to the equipment firmware version

Sub index	4
Description	Serial number
Access	RO
PDO Mapping	No
Range	UNSIGNED32
Default value	Different for every PLC300

The vendor ID is the number that identifies the manufacturer at the CiA. The product code is defined by the manufacturer according to the type of product. The revision number represents the equipment firmware version. The sub-index 4 is a unique serial number for each programmable controller PLC300 in CANopen network.

6.2 SERVICE DATA OBJECTS – SDOS

The SDOs are responsible for the direct access to the object dictionary of a specific device in the network. They are used for the configuration and therefore have low priority, since they do not have to be used for communicating data necessary for the device operation.

There are two types of SDOs: client and server. Basically, the communication initiates with the client (usually the master of the network) making a read (*upload*) or write (*download*) request to a server, and then this server answers the request.

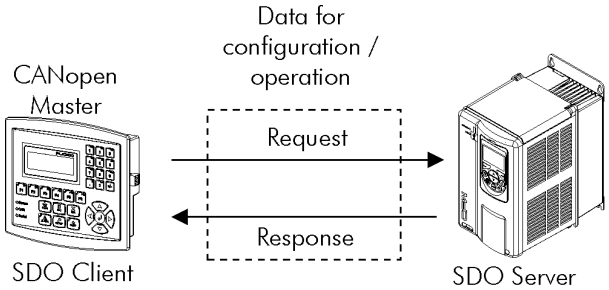


Figure 6.1: *Communication between SDO client and server*

6.2.1 Object 1200h – SDO Server

The programmable controller PLC300 working in slave mode has only one SDO of the server type, which makes it possible the access to its entire object dictionary. Through it, an SDO client can configure the communication, the parameters and the drive operation. Every SDO server has an object, of the SDO_PARAMETER type, for its configuration, having the following structure:

Index	1200h
Name	Server SDO Parameter
Object	Record
Type	SDO Parameter

Sub index	0
Description	Number of the last sub-index
Access	RO
PDO Mapping	No
Range	UNSIGNED8
Default value	2

Sub index	1
Description	COB-ID Client - Server (rx)
Access	RO
PDO Mapping	No
Range	UNSIGNED32
Default value	600h + Node-ID

Sub index	2
Description	COB-ID Server - Client (tx)
Access	RO
PDO Mapping	No
Range	UNSIGNED32
Default value	580h + Node-ID

6.2.2 SDOs Operation

A telegram sent by an SDO has an 8 byte size, with the following structure:

Identifier	8 data bytes							
11 bits	Command	Index		Sub-index	Object data			
	byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7

The identifier depends on the transmission direction (rx or tx) and on the address (or Node-ID) of the destination server. For instance, a client that makes a request to a server which Node-ID is 1, must send a message with the identifier 601h. The server will receive this message and answer with a telegram which COB-ID is equal to 581h.

The command code depends on the used function type. For the transmissions from a client to a server, the following commands can be used:

Table 6.2: *Command codes for SDO client*

Command	Function	Description	Object data
22h	Download	Write object	Not defined
23h	Download	Write object	4 bytes
2Bh	Download	Write object	2 bytes
2Fh	Download	Write object	1 byte
40h	Upload	Read object	Not used
60h or 70h	Upload segment	Segmented read	Not used

When making a request, the client will indicate through its COB-ID, the address of the slave to which this request is destined. Only a slave (using its respective SDO server) will be able to answer the received telegram to the client. The answer telegram will have also the same structure of the request telegram, the commands however are different:

Table 6.3: *Command codes for SDO server*

Command	Function	Description	Object data
60h	Download	Response to write object	Not used
43h	Upload	Response to read object	4 bytes
4Bh	Upload	Response to read object	2 bytes
4Fh	Upload	Response to read object	1 byte
41h	Upload segment	Initiates segmented response for read	4 bytes
01h ... 0Dh	Upload segment	Last data segment for read	8 ... 2 bytes

For readings of up to four data bytes, a single message can be transmitted by the server; for the reading of a bigger quantity of bytes, it is necessary that the client and the server exchange multiple telegrams.

A telegram is only completed after the acknowledgement of the server to the request of the client. If any error is detected during telegram exchanges (for instance, no answer from the server), the client will be able to abort the process by means of a warning message with the command code equal to 80h.

NOTE!
The values received from these objects are not saved in nonvolatile memory. Thus, after a shutdown or reset the equipment, the objects modified by the SDO return to its default value.

E.g.: A client SDO requests for a PLC300 at address 1 the reading of the object identified by the index 3000h, sub-index 0 (zero), which represents an 16-bit integer. The master telegram has the following format:

Identifier	Command	Index		Sub-index	Data			
601h	40h	00h	30h	00h	00h	00h	00h	00h

The PLC300 responds to the request indicating that the value of the referred object is equal to 999[4]:

Identifier	Command	Index		Sub-index	Data			
581h	4Bh	00h	30h	00h	E7	03h	00h	00h

6.3 PROCESS DATA OBJECTS – PDOS

The PDOs are used to send and receive data used during the device operation, which must often be transmitted in a fast and efficient manner. Therefore, they have a higher priority than the SDOs.

In the PDOs only data are transmitted in the telegram (index and sub-index are omitted), and in this way it is possible to do a more efficient transmission, with larger volume of data in a single telegram. However it is necessary to configure previously what is being transmitted by the PDO, so that even without the indication of the index and sub-index, it is possible to know the content of the telegram.

There are two types of PDOs, the receive PDO and the transmit PDO. The transmit PDOs are responsible for sending data to the network, whereas the receive PDOs remain responsible for receiving and handling these data. In this way it is possible to have communication among slaves of the CANopen network, it is only necessary to configure one slave to transmit information and one or more slaves to receive this information.

[4] Do not forget that for any integer type of data, the byte transfer order is from the least significant to the most significant.

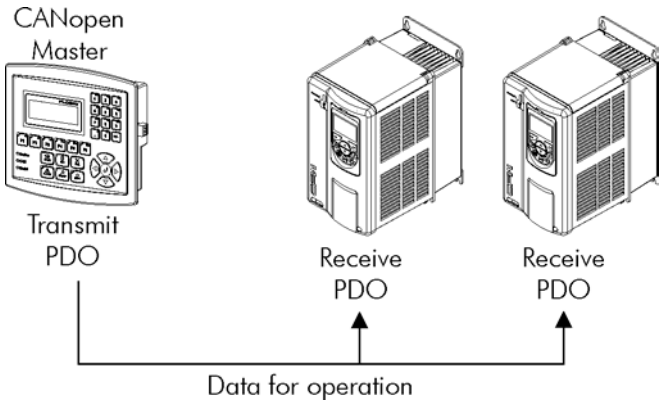


Figure 6.2: *Communication using PDOs*

NOTE!
PDOs can only be transmitted or received when the device is in the operational state. The figure 6.2 illustrates the available states for CANopen network node.

6.3.1 PDO Mapping Objects

In order to be able to be transmitted by a PDO, it is necessary that an object be mapped into this PDO content. In the description of communication objects (1000h – 1FFFh), the filed "PDO Mapping" informs this possibility. Usually only information necessary for the operation of the device can be mapped, such as enabling commands, device status, reference, etc. Information on the device configuration are not accessible through PDOs, and if it is necessary to access them one must use the SDOs.

For the manufacturer specific objects (2000h - 5FFFh), network input markers can be mapped in reception PDOs, while network output markers can be mapped to transmission PDOs. Item 5.4 describes the objects mappable to PDOs.

The EDS file brings the list of all available objects informing whether the object can be mapped or not.

6.3.2 Receive PDOs

The receive PDOs, or RPDOs, are responsible for receiving data that other devices send to the CANopen network. The programmable controller PLC300 working in slave mode has 8 receive PDOs, each one being able to receive up to 8 bytes. Each RPDO has two parameters for its configuration, a PDO_COMM_PARAMETER and a PDO_MAPPING, as described next.

PDO_COMM_PARAMETER

Index	1400h up to 1407h
Name	Receive PDO communication parameter
Object	Record
Type	PDO COMM PARAMETER

Sub index	0
Description	Number of the last sub-index
Access	ro
PDO Mapping	No
Range	UNSIGNED8
Default value	2

Sub index	1
Description	COB-ID used by the PDO
Access	rw
PDO Mapping	No
Range	UNSIGNED32
Default value	1400h: 200h + Node-ID 1401h: 300h + Node-ID 1402h: 400h + Node-ID 1403h: 500h + Node-ID 1404h – 1407h: 0

Sub index	2
Description	Transmission Type
Access	rw
PDO Mapping	No
Range	UNSIGNED8
Default value	254

The sub-index 1 contains the receive PDO COB-ID. Every time a message is sent to the network, this object will read the COB-ID of that message and, if it is equal to the value of this field, the message will be received by the device. This field is formed by an UNSIGNED32 with the following structure:

Table 6.4: *COB-ID description*

Bit	Value	Description
31 (MSB)	0	PDO is enabled
	1	PDO is disabled
30	0	RTR permitted
29	0	Identifier size = 11 bits
28 – 11	0	Not used, always 0
10 – 0 (LSB)	X	11-bit COB-ID

The bit 31 allows enabling or disabling the PDO. The bits 29 and 30 must be kept in 0 (zero), they indicate respectively that the PDO accepts remote frames (RTR frames) and that it uses an 11-bit identifier. Since the PLC300 frequency inverter does not use 29-bit identifiers, the bits from 28 to 11 must be kept in 0 (zero), whereas the bits from 10 to 0 (zero) are used to configure the COB-ID for the PDO.

The sub-index 2 indicates the transmission type of this object, according to the next table.

Table 6.5: *Description of the type of transmission*

Type of transmission	PDOs transmission				
	Cyclic	Acyclic	Synchronous	Asynchronous	RTR
0		•	•		
1 – 240	•		•		
241 – 251	Reserved				
252			•		•
253				•	•
254				•	
255				•	

- Values 0 – 240: any RPDO programmed in this range presents the same performance. When detecting a message, it will receive the data; however it won't update the received values until detecting the next SYNC telegram.
- Values 252 and 253: not allowed for receive PDOs.
- Values 254 and 255: they indicated that there is no relationship with the synchronization object. When receiving a message, its values are updated immediately.

PDO_MAPPING

Index	1600h up to 1607h
Name	Receive PDO mapping
Object	Record
Type	PDO MAPPING

Sub index	0
Description	Number of mapped objects
Access	RO
PDO Mapping	No
Range	0 = disable 1 ... 8 = number of mapped objects
Default value	0

Sub index	1 up to 8
Description	1 up to 8 object mapped in the PDO
Access	Rw
PDO Mapping	No
Range	UNSIGNED32
Default value	According EDS file

This parameter indicates the mapped objects in the PLC300 receive PDOs. It is possible to map up to 8 different objects for each RPDO, provided that the total length does not exceed eight bytes. The mapping of an object is done indicating its index, sub-index[5] and size (in bits) in an UNSIGNED32, field with the following format:

UNSIGNED32		
Index (16 bits)	Sub-index (8 bits)	Size of the object (8 bits)

For instance, analyzing the receive PDO standard mapping, we have:

- **Sub-index 0 = 2:** the RPDO has two mapped objects.
- **Sub-index 1 = 3400.0010h:** the first mapped object has an index equal to 3400h, sub-index 0 (zero), and a size of 16 bits. This object corresponds to the input marked %IW6000
- **Sub-index 2 = 3804.0020h:** the second mapped object has an index equal to 3804h, sub-index 0 (zero), and a size of 32 bits. This object corresponds to the input marked %ID6004.

It is possible to modify this mapping by changing the quantity or the number of mapped objects. Remembering that only 8 objects or 8 bytes can be mapped at maximum.

NOTE!

- In order to change the mapped objects in a PDO, it is first necessary to write the value 0 (zero) in the sub-index 0 (zero). In that way the values of the sub-indexes 1 to 8 can be changed. After the desired mapping has been done, one must write again in the sub-index 0 (zero) the number of objects that have been mapped, enabling again the PDO.
- In order to speed up the updating of data via PDO, the values received with these objects are not saved in the inverter non-volatile memory. Therefore, after switching off or resetting the equipment the objects modified by an RPDO get back to their default value.
- Do not forget that PDOs can only be received if the PLC300 is in the operational state.

6.3.3 Transmit PDOs

The transmit PDOs, or TPDOs, as the name says, are responsible for transmitting data for the CANopen network. The programmable controller PLC300 has 8 transmit PDOs, each one being able to transmit up to 8 data bytes. In a manner similar to RPDOs, each TPDO has two parameters for its configuration, a PDO_COMM_PARAMETER and a PDO_MAPPING, AS DESCRIBED NEXT.

PDO_COMM_PARAMETER

Index	1800h up to 1807h
Name	Transmit PDO Parameter
Object	Record
Type	PDO COMM PARAMETER

[5] If the object is of the VAR type and does not have sub-index, the value 0 (zero) must be indicated for the sub-index.

Sub index	0
Description	Number of the last sub-index
Access	ro
PDO Mapping	No
Range	UNSIGNED8
Default value	5

Sub index	1
Description	COB-ID used by the PDO
Access	rw
PDO Mapping	No
Range	UNSIGNED32
Default value	1800h: 180h + Node-ID 1801h: 280h + Node-ID 1802h: 380h + Node-ID 1803h: 480h + Node-ID 1804h – 1807h: 0

Sub index	2
Description	Transmission Type
Access	rw
PDO Mapping	No
Range	UNSIGNED8
Default value	254

Sub index	3
Description	Time between transmissions
Access	rw
PDO Mapping	No
Range	UNSIGNED16
Default value	-

Sub index	4
Description	Reserved
Access	rw
PDO Mapping	No
Range	UNSIGNED8
Default value	-

Sub index	5
Description	Event timer
Access	rw
PDO Mapping	No
Range	0 = disable UNSIGNED16
Default value	0

The sub-index 1 contains the transmit PDO COB-ID. Every time this PDO sends a message to the network, the identifier of that message will be this COB-ID. The structure of this field is described in table 6.4.

The sub-index 2 indicates the transmission type of this object, which follows the table 6.5 description. Its working is however different for transmit PDOs:

- Value 0: indicates that the transmission must occur immediately after the reception of a SYNC telegram, but not periodically.
- Values 1 – 240: the PDO must be transmitted at each detected SYNC telegram (or multiple occurrences of SYNC, according to the number chosen between 1 and 240).
- Value 252: indicates that the message content must be updated (but not sent) after the reception of a SYNC telegram. The transmission of the message must be done after the reception of a remote frame (RTR frame).
- Value 253: the PDO must update and send a message as soon as it receives a remote frame.
- Values 254: The object must be transmitted according to the timer programmed in sub-index 5.
- Values 255: the object is transmitted automatically when the value of any of the objects mapped in this PDO is changed. It works by changing the state (*Change of State*). This type does also allow that the PDO be transmitted according to the timer programmed in sub-index 5.

In the sub-index 3 it is possible to program a minimum time (in multiples of 100µs) that must elapse after the a telegram has been sent, so that a new one can be sent by this PDO. The value 0 (zero) disables this function.

The sub-index 5 contains a value to enable a timer for the automatic sending of a PDO. Therefore, whenever a PDO is configured as the asynchronous type, it is possible to program the value of this timer (in multiples of 1ms), so that the PDO is transmitted periodically in the programmed time.

NOTE!
- The value of this timer must be programmed according to the used transmission rate. Very short times (close to the transmission time of the telegram) are able to monopolize the bus, causing indefinite retransmission of the PDO, and avoiding that other less priority objects transmit their data.
- The minimum time allowed for this Function in the programmable controller PLC300 is 1ms.
- It is important to observe the time between transmissions programmed in the sub-index 3, especially when the PDO is programmed with the value 255 in the sub-index 2 (*Change of State*).

PDO_MAPPING

Index	1A00h up to 1A07h
Name	Transmit PDO mapping
Object	Record
Type	PDO MAPPING

Sub index	0
Description	Number of the last sub-index
Access	ro
PDO Mapping	No
Range	0 = disable 1 ... 8 = number of mapped objects
Default value	0

Sub index	1 up to 8
Description	1 up to 8 object mapped in the PDO
Access	rw
PDO Mapping	No
Range	UNSIGNED32
Default value	0

The PDO MAPPING for the transmission works in similar way than for the reception, however in this case the data to be transmitted by the PDO are defined. Each mapped object must be put in the list according to the description showed next:

UNSIGNED32		
Index (16 bits)	Sub-index (8 bits)	Size of the object (8 bits)

For instance, analyzing the standard mapping of the fourth transmit PDO, we have:

- *Sub-index 0 = 2*: This PDO has two mapped objects.
- *Sub-index 1 = 4000.0008h*: the first mapped object has an index equal to 4000h, sub-index 0 (zero), and a size of 8 bits. This object corresponds to the input marked %QB2000.
- *Sub-index 2 = 4804.0020h*: the second mapped object has an index equal to 22A9h, sub-index 0 (zero), and a size of 16 bits. This object corresponds to the input marked %QD2004.

Therefore, every time this PDO transmits its data, it elaborates its telegram containing four data bytes, with the values of the parameters P0680 and P0681. It is possible to modify this mapping by changing the quantity or the number of mapped objects. Remember that a maximum of 8 objects or 8 bytes can be mapped.

NOTE!
In order to change the mapped objects in a PDO, it is first necessary to write the value 0 (zero) in the sub-index 0 (zero). In that way the values of the sub-indexes 1 to 8 can be changed. After the desired mapping has been done, one must write again in the sub-index 0 (zero) the number of objects that have been mapped, enabling again the PDO.

6.4 SYNCHRONIZATION OBJECT – SYNC

This object is transmitted with the purpose of allowing the synchronization of events among the CANopen network devices. It is transmitted by a SYNC producer, and the devices that detect its transmission are named SYNC consumers

The programmable controller PLC300 working in slave mode has the function of a SYNC consumer and, therefore, it can program its PDOs to be synchronous. As described in table 6.5, synchronous PDOs are those related to the synchronization object, thus they can be programmed to be transmitted or updated based in this object.

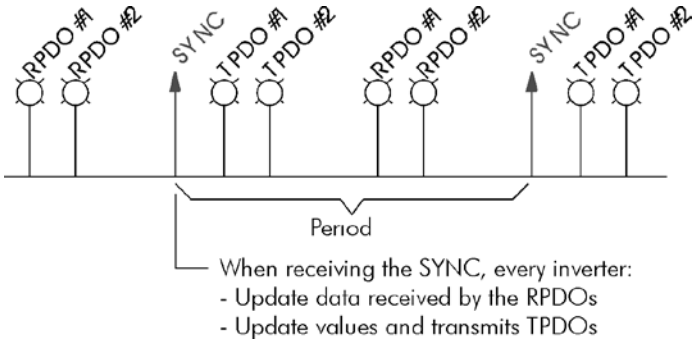


Figure 6.3: *SYNC*

The SYNC message transmitted by the producer does not have any data in its data field, because its purpose is to provide a time base for the other objects. There is an object in the PLC300 for the configuration of the COB-ID of the SYNC consumer.

Index	1015h
Name	COB-ID SYNC
Object	VAR
Type	UNSIGNED32

Access	rw
PDO Mapping	No
Range	UNSIGNED32
Default value	80h

NOTE!
The period of the SYNC telegrams must be programmed in the producer according to the transmission rate and the number of synchronous PDOs to be transmitted. There must be enough time for the transmission of these objects, and it is also recommended that there is a tolerance to make it possible the transmission of asynchronous messages, such as EMCY, asynchronous PDOs and SDOs.

6.5 NETWORK MANAGEMENT – NMT

The network management object is responsible for a series of services that control the communication of the device in a CANopen network. For the PLC300 the services of node control and error control are available (using *Node Guarding* or *Heartbeat*).

6.5.1 Slave State Control

With respect to the communication, a CANopen network device can be described by the following state machine:

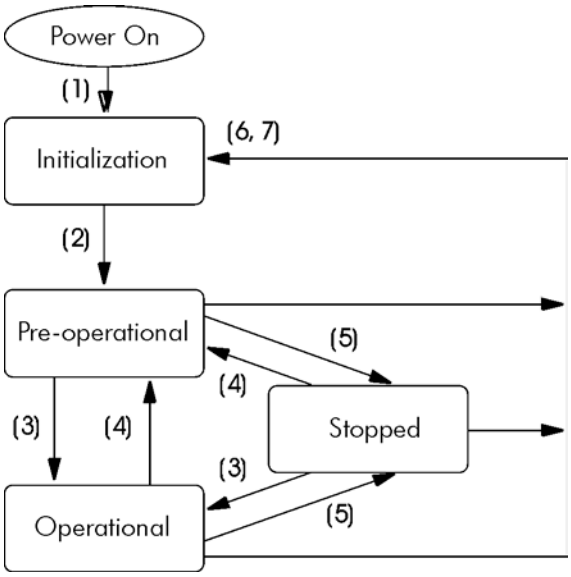


Figure 6.4: *CANopen node state diagram*

Table 6.6: *Transitions Description*

Transition	Description
1	The device is switched on and initiates the initialization (automatic).
2	Initialization concluded, it goes to the preoperational state (automatic).
3	It receives the Start Node command for entering the operational state.
4	It receives the Enter Pre-Operational command, and goes to the preoperational state.
5	It receives the Stop Node command for entering the stopped state.
6	It receives the Reset Node command, when it executes the device complete reset.
7	It receives the Reset Communication command, when it reinitializes the object values and the CANopen device communication.

During the initialization the Node-ID is defined, the objects are created and the interface with the CAN network is configured. Communication with the device is not possible during this stage, which is concluded automatically. At the end of this stage the slave sends to the network a telegram of the Boot-up Object, used only to indicate that the initialization has been concluded and that the slave has entered the preoperational state. This telegram has the identifier 700h + Node-ID, and only one data byte with value equal to 0 (zero).

In the preoperational state it is already possible to communicate with the slave, but its PDOs are not yet available for operation. In the operational state all the objects are available, whereas in the stopped state only the NMT object can receive or transmit telegrams to the network. The next table shows the objects available for each state.

Table 6.7: *Objects accessible in each state*

	Initialization	Preoperational	Operational	Stopped
PDO			•	
SDO		•	•	
SYNC		•	•	
EMCY		•	•	
Boot-up	•			
NMT		•	•	•

This state machine is controlled by the network master, which sends to each slave the commands so that the desired state change be executed. These telegrams do not have confirmation, what means that the slave does only receive the telegram without returning an answer to the master. The received telegrams have the following structure:

Identifier	byte 1	byte 2
00h	Command code	Destination Node-ID

Table 6.8: *Commands for the state transition*

Command code	Destination Node-ID
1 = START node (transition 3) 2 = STOP node (transition 4) 128 = Enter pre-operational (transition 5) 129 = Reset node (transition 6) 130 = Reset communication (transition 7)	0 = All the slaves 1 ... 127 = Specific slave

The transitions indicated in the command code correspond to the state transitions executed by the node after receiving the command (according to the Figure 6.4). The *Reset node* command makes the PLC300 execute a complete reset of the device, while the *Reset communication* command causes the device to reinitialize only the objects pertinent to the CANopen communication.

6.5.2 Error Control – Node Guarding

This service is used to make it possible the monitoring of the communication with the CANopen network, both by the master and the slave as well. In this type of service the master sends periodical telegrams to the slave, which responds to the received telegram. If some error that interrupts the communication occurs, it will be possible to identify this error, because the master as well as the slave will be notified by the *Timeout* in the execution of this service. The error events are called *Node Guarding* for the master and *Life Guarding* for the slave.

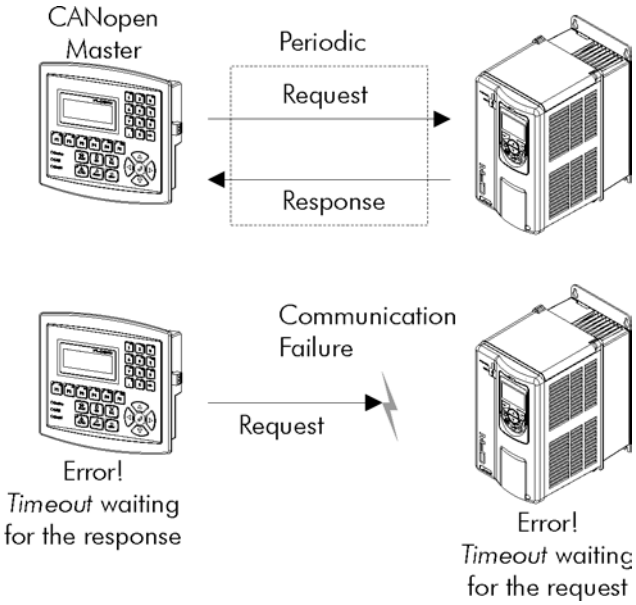


Figure 6.5: *Error control service – Node Guarding*

There are two objects of the dictionary for the configuration of the error detection times for the *Node Guarding* service:

Index	100Ch
Name	Guard Time
Object	VAR
Type	UNSIGNED16

Access	rw
PDO Mapping	No
Range	UNSIGNED16
Default value	0

Index	100Dh
Name	Life Time Factor
Object	VAR
Type	UNSIGNED8

Access	rw
PDO Mapping	No
Range	UNSIGNED8
Default value	0

The 100Ch object allows programming the time necessary (in milliseconds) for a fault occurrence being detected, in case the PLC300 does not receive any telegram from the master. The 100Dh object indicates how many faults in sequence are necessary until it be considered that there was really a communication error. Therefore, the multiplication of these two values will result in the total necessary time for the communication error detection using this object. The value 0 (zero) disables this function.

Once configured, the PLC300 starts counting these times starting from the first *Node Guarding* telegram received from the network master. The master telegram is of the remote type, not having data bytes. The identifier is equal to 700h + Node-ID of the destination slave. However the slave response telegram has 1 data byte with the following structure:

Identifier	byte 1	
	bit 7	bit 6 ... bit 0
700h + Node-ID	Toggle	Slave state

This telegram has one single data byte. This byte contains, in the seven least significant bits, a value to indicate the slave state (4 = stopped, 5 = operational and 127 = preoperational), and in the eighth bit, a value that must be changed at every telegram sent by the slave (*toggle bit*).

If the programmable controller PLC300 working in slave mode detects an error using this mechanism, it will turn automatically to the preoperational state and indicate this state in the LED CAN.

NOTE!

- This object is active even in the stopped state (see table 6.7).
- The value 0 (zero) in any of these two objects will disable this function.
- If after the error detection the service is enabled again, then the error indication will be removed from the HMI.
- The minimum value accepted by the PLC300 is 1ms., but considering the transmission rate and the number of nodes in the network, the times programmed for this function must be consistent, so that there is enough time for the transmission of the telegrams and also that the rest of the communication be able to be processed.
- For any every slave only one of the two services - Heartbeat or Node Guarding – can be enabled.

6.5.3 Error Control – Heartbeat

The error detection through the *Heartbeat* mechanism is done using two types of objects: the *Heartbeat* producer and the *Heartbeat* consumer. The producer is responsible for sending periodic telegrams to the network, simulating a heartbeat, indicating that the communication is active and without errors. One or more consumers can monitor these periodic telegrams, and if they cease occurring, it means that any communication problem occurred.

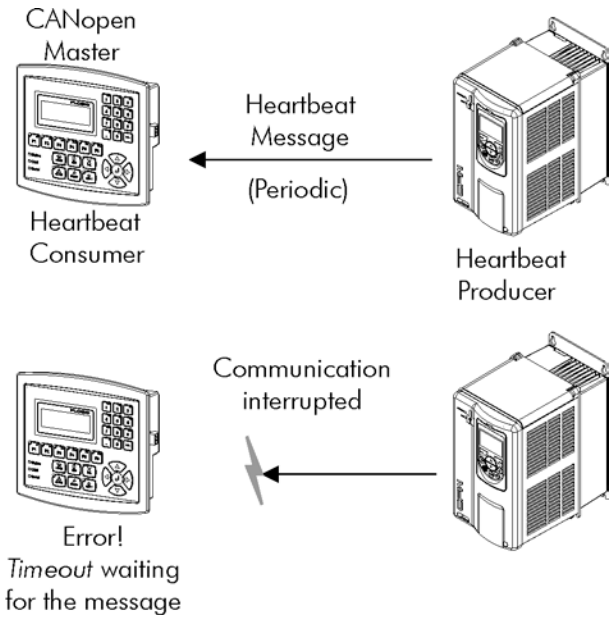


Figure 6.6: *Error control service – Heartbeat*

One device of the network can be both producer and consumer of *heartbeat* messages. For example, the network master can consume messages sent by a slave, making it possible to detect communication problems with the master, and simultaneously the slave can consume *heartbeat* messages sent by the master, also making it possible to the slave detect communication fault with the master.

The PLC300 has the producer and consumer of *heartbeat* services. As a consumer, it is possible to program up to 4 different producers to be monitored by the inverter.

Index	1016h
Name	Consumer Heartbeat Time
Object	ARRAY
Type	UNSIGNED32

Sub index	0
Description	Number of the last sub-index
Access	ro
PDO Mapping	No
Range	-
Default value	63

Sub index	1 – 63
Description	Consumer Heartbeat Time 1 – 63
Access	rw
PDO Mapping	No
Range	UNSIGNED32
Default value	0

At sub-indexes 1 to 63, it is possible to program the consumer by writing a value with the following format:

UNSIGNED32		
Reserved (8 bits)	Node-ID (8 bits)	Heartbeat time (16 bits)

- Node-ID: it allows programming the Node_ID for the *heartbeat* producer to be monitored.
- *Heartbeat time*: it allows programming the time, in 1 millisecond multiples, until the error detection if no message of the producer is received. The value 0 (zero) in this field disables the consumer.

Once configured, the *heartbeat* consumer initiates the monitoring after the reception of the first telegram sent by the producer. In case that an error is detected because the consumer stopped receiving messages from the *heartbeat* producer, the programmable controller will turn automatically to the preoperational state and indicate this state in the LED CAN.

As a producer, the programmable controller PLC300 has an object for the configuration of that service:

Index	1017h
Name	Producer Heartbeat Time
Object	VAR
Type	UNSIGNED16

Access	rw
PDO Mapping	No
Range	UNSIGNED8
Default value	0

The 1017h object allows programming the time in milliseconds during which the producer has to send a *heartbeat* telegram to the network. Once programmed, the inverter initiates the transmission of messages with the following format:

Identifier	byte 1	
	bit 7	bit 6 ... bit 0
700h + Node-ID	Always 0	Slave state

NOTE!

- This object is active even in the stopped state (see table 6.7).
- The value 0 (zero) in the object will disable this function.
- If after the error detection the service is enabled again, then the error indication will be removed from the HMI.
- The time value programmed for the consumer must be higher than the programmed for the respective producer. Actually, it is recommended to program the consumer with a multiple of the value used for the producer.
- For any every slave only one of the two services - *Heartbeat* or *Node Guarding* – can be enabled.

6.6 INITIALIZATION PROCEDURE

Once the operation of the objects available for the programmable controller PLC300 is known, then it becomes necessary to program the different objects to operate combined in the network. In a general manner, the procedure for the initialization of the objects in a CANopen network follows the description of the next flowchart:

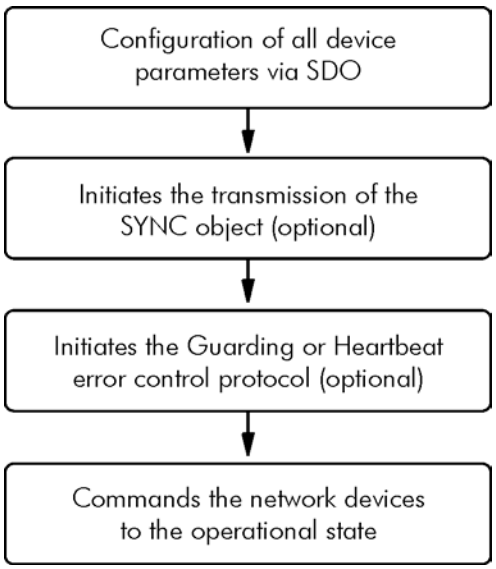


Figure 6.7: *Initialization process flowchart*

It is necessary to observe that the programmable controller PLC300 communication objects (1000h to 1FFFh) are not stored in the nonvolatile memory. Therefore, every time the equipment is reset or switched off, it is necessary to redo the communication objects parameter setting.

7 OPERATION IN CANOPEN NETWORK – MASTER MODE

In addition to operating as a salve, the programmable controller PLC300 can also operate as master of the CANopen network. Below are described the characteristics and functions of the PLC300 as master of the CANopen network.

7.1 ENABLING OF THE MASTER CANOPEN FUNCTION

As default, the programmable controller PLC300 is programmed to operate as slave of the CANopen network. The programming of the equipment as network master must be done by using the WPSCAN software, which also allows the configuration of the entire CANopen network. The detailed description of the windows and functions of the WPSCAN software is obtained in the "Help" menu of the software itself.

After the configuration of the master is ready, it is necessary to download[6] the configurations via one of the programming interfaces of the product – refer to the user's manual for further information. Once set as network master, if necessary to erase those configurations, the function to erase the user's program – trough Setup menu – also erases the configurations of the CANopen master.

NOTE!
The CANopen network is a flexible network that allows several forms of configurations and operation. However, in order to use this flexibility, it is necessary that the user know well both the communication functions and objects used to configure the network, and the WPSCAN programming software.

7.2 CHARACTERISTICS OF THE CANOPEN MASTER

The programmable controller PLC300 allows controlling a group of up to 63 slaves, using the following communication services and resources:

- Network manager task (NMT)
- 63 transmission PDOs
- 63 reception PDOs
- 63 Heartbeat Consumers
- Heartbeat Producer
- SDO Client
- SYNC producer/consumer
- 512 bytes of network input markers
- 512 bytes of network output markers

The physical characteristics – installation, connector, cable, etc. – are the same for the PLC300 operating as both master and slave. The configurations of address and baud rate are also necessary to operate as master, but these configurations are programmed by the WPSCAN software according to the properties defined for the master in the software itself.

NOTE!
The network input markers are used to map data in RPDOs, while the network output markers are used to map data in TPDOs. They can be accessed in Byte (IB% or% QB), Word (% IW or% QW) or Double Word (% ID or% DR). Its function, however, is not pre-defined and depends on the ladder application developed for the PLC300.

7.3 OPERATION OF THE MASTER

Once programmed to operate as master, the programmable controller PLC300 will execute the following steps to initialize, in a sequence, each slave:

- 1st: send the communication reset command to the entire network, so that the slaves initialize with known values for the communication objects.

6 During the download of the configurations, the CANopen communication will be disabled, and it will be enabled again at the end of the operation.

- 2nd: Identification of the equipment in network, trough the reading via SDO of the object 1000h/00h – Object Identification.
- 3rd: Writing via SDO of all the objects programmed for the slave, which usually includes the configuration and mapping of the TPDOs and RPDOs, node guarding, heartbeat, besides the specific objects of the manufacturer, in case they are programmed.
- 4th: Start the error control task – node guarding or heartbeat – if they are programmed.
- 5th: send the slave to mode of operation.

If one of these steps fails, the error of communication with slave will occur. Depending of the configurations, the slave initialization will be aborted, and the master will initialize the next slave, returning to the slave with error after trying to initialize all the other network slaves.

Similarly, if, during the operation of a slave, an error is identified in the error control task, depending on the configurations of the master, the slave will be automatically reset and the initialization procedure will be run again.

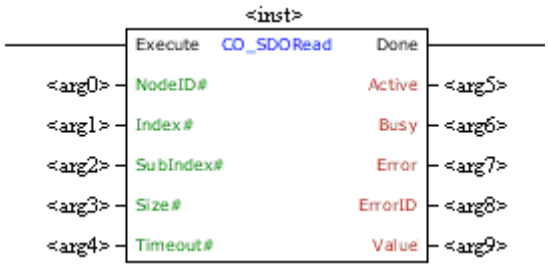
NOTE!
The communication status and the status of each slave can be observed in system markers.

7.4 BLOCKS FOR THE CANOPEN MASTER

In addition to the communication objects and the configurations made on the WSCAN software, blocks for monitoring and sending commands are also available. They can be used during the preparation of the ladder application for the programmable controller PLC300. It is not necessary to use these blocks during the equipment operation, but they provides more flexibility and simplify the communication troubleshooting during the operation of the programmable controller PLC300.

7.4.1 CANopen SDO Read

Block for data reading via SDO of a remote slave. It allows the reading of objects in the network with a size of up to 4 bytes.



It has an "Execute" block enabling input, and a "Done" output which is activated after the end of the function's successful performance. In the "Execute" positive transition, when the master's SDO client is free, a new requisition is sent to the slave's SDO server. At the operation successful end – response received from the slave – the "Done" output is activated, remaining active while the input is active. In case of error in the requisition performance, the "Error" output is enabled, and the error code is put to "ErrorID".

<inst> - insert an instance (tag).

<arg0>: "NodeID#" - VAR_IN: insert a constant.
Types of data BYTE
Description: Address of destination slave - 1 to 127

<arg1>: "Index#" - VAR_IN: insert a constant.
Types of data WORD
Description: Index of object accessed, among the objects available in the slave's dictionary of objects - 0 to 65535.

<arg2>: "SubIndex#" - VAR_IN: insert a constant.
Types of data BYTE
Description: Sub-index of the accessed object - 0 to 255.

<arg3>: "Size#" - VAR_IN: insert a constant.
Types of data BYTE
Description: Size of the accessed data in bytes - 1 to 4.

<arg4>: "Timeout#" - VAR_IN: insert a constant.
Types of data WORD
Description: Waiting time for the arrival of the response by the slave, from its sending by the master - 5 to 5000 ms.

<arg5>: "Active" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: Active block, request for reading sent to the slave and awaiting response.
Note: The variable must have writing permission.

<arg6>: "Busy" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: Block enabled, though resource is not available (SDO client sending another requisition), waiting for release so that the request is sent by the block. If the enabling input is removed while the block makes that indication, the requisition is rejected.
Note: The variable must have writing permission.

<arg7>: "Error" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: error during requisition performance.
Note: The variable must have writing permission.

<arg8>: "ErrorID" - VAR_OUT: insert a variable (tag).
Types of data BYTE or USINT.
Description: In case of error during the requisition, it indicates the type of error occurred. Possible results: 0= "Successfully performed"; 1= "Card cannot perform the function" (example: Master disabled); 2= "Timeout in the response by the slave"; 3= "Slave returned error".
Note: The variable must have writing permission.

<arg9>: "Value" - VAR_OUT: insert a variable (tag).
Types of data BYTE[1 ... 4] or USINT[1 ... 4]
Description: Variable or array where the slave's read data will be saved
Note: The variable must have writing permission.

7.4.2 CANopen SDO Write

Block for data writing via SDO of a remote slave. It allows the writing of objects in the network with the size of up to 4 bytes.

It has an "Execute" block enabling input, and a "Done" output which is activated after the end of the function's successful performance. In the "Execute" positive transition, when the master's SDO client is free, a new requisition is sent to the slave's SDO server. At the operation successful end – response received from the slave

– the "Done" output is activated, remaining active while the input is active. In case of error in the requisition performance, the "Error" output is enabled, and the error code is put to "ErrorID".

<inst> - insert an instance (tag).

<arg0>: "NodeID#" - VAR_IN: insert a constant.
Types of data BYTE
Description: Address of destination slave - 1 to 127

<arg1>: "Index#" - VAR_IN: insert a constant.
Types of data WORD
Description: Index of the accessed object, among the objects available in the slave's dictionary of objects - 0 to 65535.

<arg2>: "SubIndex#" - VAR_IN: insert a constant.
Types of data BYTE
Description: Sub-index of the accessed object - 0 to 255.

<arg3>: "Size#" - VAR_IN: insert a constant.
Types of data BYTE
Description: Size of the accessed data in bytes - 1 to 4.

<arg4>: "Timeout#" - VAR_IN: insert a constant.
Types of data WORD
Description: Waiting time for the arrival of the response by the slave, from the sending by the master - 5 to 5000 ms.

<arg5>: "Value" - VAR_IN: insert a variable (tag).
Types of data BYTE[1 ... 4] or USINT[1 ... 4]
Description: Variable or array with data to send to the slave

<arg6>: "Active" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: Active block, requisition for reading sent to the slave and awaiting response.
Note: The variable must have writing permission.

<arg7>: "Busy" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: Block enabled, though resource is not available (SDO client sending another requisition), waiting for release so that the request is sent by the block. If the enabling input is removed while the block makes that indication, the requisition is rejected.
Note: The variable must have writing permission.

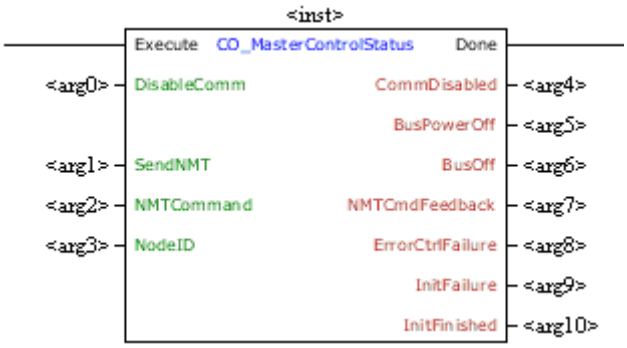
<arg8>: "Error" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: error during requisition performance.
Note: The variable must have writing permission.

<arg9>: "ErrorID" - VAR_OUT: insert a variable (tag).
Types of data BYTE or USINT.
Description: In case of requisition error, the type of error occurred will be indicated. Possible results: 0= "Successfully performed"; 1= "Card cannot perform the function" (example: Master disabled); 2= "Timeout in the response by the slave"; 3= "Slave returned error".
Note: The variable must have writing permission.

NOTE!
- It is important that the quantity of read or written data programmed in the blocks is compatible with the size of the variable, or the array with the value,
- In case of error returned by the slave, it is possible to obtain the code of the last error received through the reading system markers. Refer to item 8 for a list of available markers.

7.4.3 CANopen Master Control/Status

Block to control and monitor the master in the CANopen network. It shows the state of the network master for diagnosis and identification of communication problems, as well as allows the sending of commands to the network management task – NMT.



It has an "Execute" block enabling input, and a "Done" output which is activated after the end of the function performance. If the "Execute" input is active, it updates the values of inputs and outputs and enables the "Done" output. If the "Execute" input is not active, the other input values are ignored and all outputs are zeroed.

<inst> - insert an instance (tag).

<arg0>: "DisableComm" - VAR_IN: insert a constant or a variable (tag).
Types of data BOOL
Description: Disables the CANopen communication. When disabling the master, the CANopen master's status counters and markers are also zeroed - 0 or 1.

<arg1>: "SendNMT" - VAR_IN: insert a constant or a variable (tag).
Types of data BOOL
Description: During the transition of this signal, the CANopen master triggers the sending of a management command - NMT - according to the command and the address programmed in this block - 0 or 1.

<arg2>: "NMTCommand" - VAR_IN: insert a constant or a variable (tag).
Types of data BYTE
Description: It indicates which command must be sent to the slave: 1= "Start node"; 2= "Stop node"; 128= "Enter pre-operational"; 129= "Reset node"; 130= "Reset communication".

<arg3>: "NodeID" - VAR_IN: insert a constant or a variable (tag).
Types of data BYTE or USINT.
Description: Slave's address for the sending of the NMT command - 0= Broadcast (message to all slaves); 1 to 127= Slave's specific address.

<arg4>: "CommDisabled" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master and the communication in the CAN interface were disabled. It is indicated whenever the user command to disable the interface is received, but it is also indicated in those situations of lack of power supply in the CAN interface or bus off: 0= "Communication Enabled"; 1= "Communication Disabled".
Note: The variable must have writing permission.

<arg5>: "BusPowerOff" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that failure in the CAN interface power supply was detected: 0= "Interface CAN supplied"; 1= "Interface CAN without power supply".
Note: The variable must have writing permission.

<arg6>: "BusOff" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that bus off error was detected in the CAN interface: 0= "Without bus off error"; 1= "With bus off error".
Note: The variable must have writing permission.

<arg7>: "NMTCmdFeedback" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the management command was sent by the master: 0= "Without command or command not sent"; 1= "NMT command sent".
Note: The variable must have writing permission.

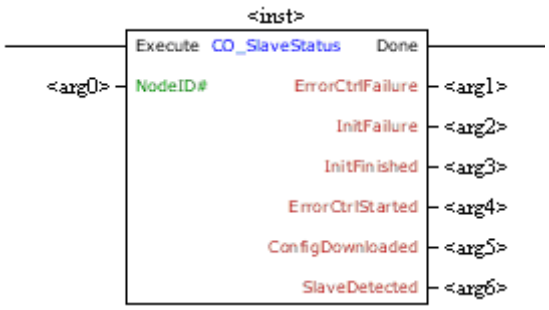
<arg8>: "ErrorCtrlFailure" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master has detected error in the error control task (node guarding or heartbeat) in at least one slave in the network: 0= "Without detected error"; 1= "Master detected error in the node guarding or heartbeat in at least one slave in the network".
Note: The variable must have writing permission.

<arg9>: "InitFailure" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master has detected error during the initialization of at least one slave in the network: 0= "Without detected error"; 1= "Master detected error in the initialization in at least one slave in the network".
Note: The variable must have writing permission.

<arg10>: "InitFinished" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master has tried to initialize all slaves in the network. The initialization was not necessarily performed successfully; there might have been errors during initialization: 0= "Master has not yet performed the initialization procedure of all slaves"; 1= "Master carried out the initialization (successfully or unsuccessfully) of all slaves".
Note: The variable must have writing permission.

7.4.4 CANopen Slave Status

Block to monitor the slave of the CANopen network. It shows the state of a slave in the network for diagnosis and identification of communication problems.



It has an "Execute" block enabling input, and a "Done" output which is activated after the end of the function's successful performance. If the "Execute" input is active, it updates the values of inputs and outputs and enables the "Done" output. If the "Execute" input is not active, the other input values are ignored and all outputs are cleared.

<inst> - insert an instance (tag).

<arg0>: "NodeID" - VAR_IN: insert a constant or a variable (tag).
Types of data BYTE or USINT.
Description: Slave's address to identify the state of the communication with the master - 1 to 127.

<arg1>: "ErrorCtrlFailure" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master has detected error in the error control task (node guarding or heartbeat) in the indicated slave: 0= "Without detected error"; 1= "Master detected error in the node guarding or heartbeat in the slave".
Note: The variable must have writing permission.

<arg2>: "InitFailure" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master has detected error during the initialization of the indicated slave: 0= "Without detected error"; 1= "Master detected error in the slave initialization".
Note: The variable must have writing permission.

<arg3>: "InitFinished" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master performed the complete and successful initialization of the indicated slave: 0= "Master did not conclude the slave initialization procedure"; 1= "Master successfully performed the slave initialization".
Note: The variable must have writing permission.

<arg4>: "ErrprCtrlStarted" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master has started the error control task (node guarding or heartbeat) with the indicated slave: If this task is not enabled for the slave, this bit will be activated after performing the configuration: 0= "Error control with the slave not started"; 1= "Error control with the slave started".
Note: The variable must have writing permission.

<arg5>: "ConfigDownloaded" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master successfully finished downloading the configurations via SDO to the indicated slave: 0= "Master did not finish downloading the configurations to the slave"; 1= "Download of configurations to the slave successfully finished".
Note: The variable must have writing permission.

<arg6>: "SlaveDetected" - VAR_OUT: insert a variable (tag).
Types of data BOOL
Description: It indicates that the master was able to read the identification via the indicated slave SDO: 0= "Slave has not been contacted"; 1= "Slave successfully contacted".
Note: The variable must have writing permission.

NOTE!
The data accessed through the use of this block is also available through reading and writing system markers, as described in item 8.

8 CAN/CANOPEN SYSTEM MARKERS

For CAN interfaces, the following reading system markers (%S) and writing system markers (%C) were provided for control and monitoring:

8.1 READING SYSTEM MARKERS

CAN Interface Status: group of reading markers to indicate information about the CAN interface status.	
Marker	Description
%SB3150	CAN interface status: 0 = initializing 1 = Reserved 2 = CAN Enabled 3 = Warning 4 = Error Passive 5 = Bus Off 6 = No bus power
%SB3151	CAN bus power. 0 = No bus power 1 = Bus power detected
%SW3152	Number of received telegrams. It resets every time it reaches the maximum or when the interface is disabled.
%SW3154	Number of transmitted telegrams. It resets every time it reaches the maximum or when the interface is disabled.
%SW3156	Bus Off error counter.
%SW3158	CAN lost messages counter (overrun).

CANopen Communication Status: read markers to indicate information about the state of the CANopen communication.	
Marker	Description
%SB3180	CANopen protocol state. Indicates whether the interface is operating correctly and, in the case of the network operation as a slave, if any error was detected in the error detection mechanisms of the CANopen protocol: 0 = Disabled 1 = Reserved 2 = CANopen enabled 3 = Controle de erros enabled 4 = Node guarding error 5 = Heartbeat error
%SB3181	CANopen node state, according to the slave state machine of CANopen network: 0 = Initializing 1 = Stopped 2 = Operational 3 = Pre-operational

State of CANopen Master and Slaves: group of reading markers to indicate information about the general state of the CANopen master and the communication state between the master and each slave.	
Marker	Description
%SW3200	CANopen master state: Bit 0: all slaves have been contacted. Bit 1: download of slaves configuration done. Bit 2: error control mechanism for slaves initiated. Bit 3: slaves initialization finished. Bit 4: error detected during initialization of at least one slave. Bit 5: error detected ate error control mechanism of at least one slave. Bits 6 e 7: reserved Bit 8: assumes the value of the toogle bit (see %CD3120) after the master sending a NMT command. Bits 9 ... 12: reserved Bit 13: bus off. Bit 14: no bus power supply. Bit 15: communication disabled.
%SW3202 ... %SW3454	CANopen slaves state. There are 127 markers, each marker is associated with an address in the CANopen network, and indicates the status of the slave at address: Bit 0: slave successful contacted. Bit 1: slave configuration downloaded successfully. Bit 2: error control initiated. Bit 3: slave initialization finished. Bit 4: error during slave initialization. Bit 5: error control mechanism detected communication failure. Bits 6 ... 15: reserved

Last Error at SDO Client: group of reading markers to report data errors detected by the SDO client. If SDO client makes any request and the slave does not respond, or respond with an error, the data for the last error detected by the SDO client are saved in these markers.	
Marker	Description
%SW3460	Slave address destination for which the SDO request was sent.
%SW3462	Index of accessed object via SDO.
%SW3464	Sub-index of accessed object.
%SW3466	Type of access: 1 = read, 2 = write.
%SD3468	For writing access, indicates the written value.
%SD3472	Indicates the received error code, according to communication errors via SDO of the CANopen protocol specification.

Last detected EMCY: group of reading markers to inform about errors reported by EMCY producers. The CANopen master controller does not have EMCY consumer. EMCY telegrams sent by the network slaves, however, are captured by the master, and the information of the last EMCY detected is saved in these markers.	
Marcador	Description
%SB3480	EMCY slave address.
%SB3481	Reserved.
%SB3482 ... %SB3489	Eight data bytes of EMCY telegram, with information about the error code reported by the slave.

8.2 WRITING SYSTEM MARKERS

CAN Interface Configuration: group of writing markers to program the settings of the CAN interface. They are also accessible via the Setup menu.	
Marker	Description
%CB3052	CANopen address (Node ID). Valid range 1 to 127.
%CB3053	Reserved.
%CB3054	Reserved.
%CB3055	CAN baud rate: 0 = 1 Mbit/s 1 = reserved 2 = 500 Kbit/s 3 = 250 Kbit/s 4 = 125 Kbit/s 5 = 100 Kbit/s 6 = 50 Kbit/s 7 = 20 Kbit/s

CANopen Master Control: group of writing markers to control the CANopen master.	
Marker	Description
%CD3120	Command to control the CANopen master and to send NMT telegram. Bits 0 ... 7: NMT command code: 1 = START 2 = STOP 128 = ENTER PRE-OPERATIONAL 129 = RESET NODE 130 = RESET COMMUNICATION Bit 8: toggle bit; master sends the programmed command whenever the value of this bit changes. Bits 9 ... 14: reserved Bit 15: disables the CANopen communication Bits 16 ... 23: destination slave address for sending NMT command. Bits 24 ... 31: reserved

9 FAULTS AND ALARMS RELATED TO THE CANOPEN COMMUNICATION

CAN INTERFACE WITHOUT POWER SUPPLY

Description:
It indicates that the CAN interface does not have power supply between the pins 1 and 5 of the connector.

Actuation:
In order that it be possible to send and receive telegrams through the CAN interface, it is necessary to supply external power to the interface circuit.

If the CAN interface is connected to the power supply and the absence of power is detected, this will be indicated at CAN LED and CAN markers. If the circuit power supply is reestablished, the CAN communication will be reinitiated.

Possible Causes/Correction:
- Measure the voltage between the pins 1 and 5 of the CAN interface connector.
- Verify if the power supply cables have not been changed or inverted.
- Make sure there is no contact problem in the cable or in the CAN interface connector.

BUS OFF

Description:
The *bus off* error in the CAN interface has been detected.

Actuation:
If the number of reception or transmission errors detected by the CAN interface is too high[7], the CAN controller can be taken to the *bus off* state, where it interrupts the communication and disables the CAN interface.

In this case this will be indicated at CAN LED and CAN markers. In order that the communication be reestablished, it will be necessary to cycle the power of the product, or remove the power supply from the CAN interface and apply it again, so that the communication be reinitiated.

Possible Causes/Correction:
- Verify if there is any short-circuit between the CAN circuit transmission cables.
- Verify if the cables have not been changed or inverted.
- Verify if all the network devices use the same baud rate.
- Verify if termination resistors with the correct values were installed only at the extremes of the main bus.
- Verify if the CAN network installation was carried out in proper manner.

NODE GUARDING/HEARTBEAT

Description:
The CANopen communication error control detected a communication error by using the guarding mechanism.

Operation:
By using the error control mechanisms – Node Guarding or Heartbeat – the master and the slave can exchange periodic telegrams, with a predetermined period. If the communication is interrupted by some reason, the master, as well as the slave, will be able to detect communication error through the timeout in the exchange of those messages.

In this case this will be indicated at CAN LED and CAN markers.

[7] For more information on the error detection, refer to the CAN specification.

Possible Causes/Correction:

- Verify the times programmed in both master and slave, for the message exchanging. In order to avoid problems due to transmission delays and differences in the time counting, it is recommended that the values programmed for message exchanging in the master be a little bit shorter than the times programmed for the error detection by the slave.
- Verify if the master is sending the *guarding* telegrams in the programmed time.
- Verify communication problems that can cause telegram losses or transmission delays.

WEG Equipamentos Elétricos S.A.
Jaraguá do Sul – SC – Brasil
Fone 55 (47) 3276-4000 – Fax 55 (47) 3276-4020
São Paulo – SP – Brasil
Fone 55 (11) 5053-2300 – Fax 55 (11) 5052-4212
automacao@weg.net
www.weg.net