SoMachine

CANopen Management Functions
CAA CiA 405 Library Guide

06/2011

EIO0000000316.02

www.schneider-electric.com

Schneider Electric

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

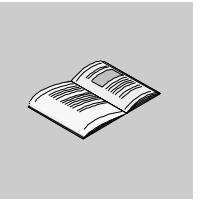
All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

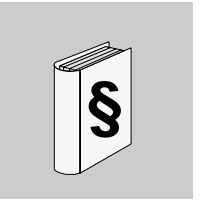
Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

Table of Contents



Safety Information



Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of this symbol to a Danger or Warning safety label indicates that an electrical hazard exists, which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates an imminently hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a potentially hazardous situation which, if not avoided, **can result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a potentially hazardous situation which, if not avoided, **can result in** minor or moderate injury.

CAUTION

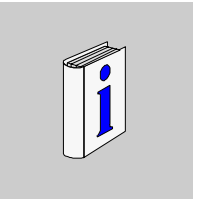
CAUTION, used without the safety alert symbol, indicates a potentially hazardous situation which, if not avoided, **can result in** equipment damage.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

About the Book



At a Glance

Document Scope

This document describes the SoMachine CAA CiA 405 Library function blocks used to manage and monitor the CANopen network and devices from the controller application.

Validity Note

This document has been updated with the release of SoMachine V3.0.

Product Related Information

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Schneider Electric for use with this equipment.
- Update your application program every time you change the physical hardware configuration.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

⚠ WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop, power outage and restart.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.
- Observe all accident prevention regulations and local safety guidelines.[1]
- Each implementation of this equipment must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[1] For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control" and to NEMA ICS 7.1 (latest edition), "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" or their equivalent governing your particular location.

User Comments

We welcome your comments about this document. You can reach us by e-mail at techcomm@schneider-electric.com.

Introduction

I

Overview

This part gives an overview of the various protocols used in CANopen, and introduces the CANopen interface and device profile for IEC 61131-3 programmable controllers (CiA405)

NOTE: Some content of this introduction has been extracted and adapted from the CAN in Automation (CiA) web site (www.can-cia.org). CiA® and CANopen® are registered Community Trademarks of CAN in Automation e.V

About CiA: CAN in Automation (CiA) is the international users and manufacturers group for Controller Area Network (CAN). The non-profit association provides technical, product and marketing information about CAN, internationally standardized in the ISO 11898 series. The aim is to promote the image of CAN and to provide a path for future developments of the CAN technology. An important part of the effort of organization is to develop and maintain the CANopen specifications as well as the support of all other internationally standardized CAN-based higher-layer protocols.

What's in this Part?

This part contains the following chapters:

Introduction to CANopen Protocol

1

What's in this Chapter?

This chapter contains the following topics:

Introduction

CANopen Protocol

The CANopen protocol provides standardized communication objects (COB) for real-time data (process data objects (PDO)), configuration data (service data objects (SDO)), and network management data (boot-up message, NMT message, and error control message) as well as other functions (time stamp, SYNC message, emergency message). All communication objects are accessible through the CAN network in the device object dictionary. These objects are addressable by a 16-bit index. In the case of array and record objects, there is an additional 8-bit sub-index. A communication object is identified by a dedicated and unique identifier (COB ID) over the CANopen network.

Device Object Dictionary

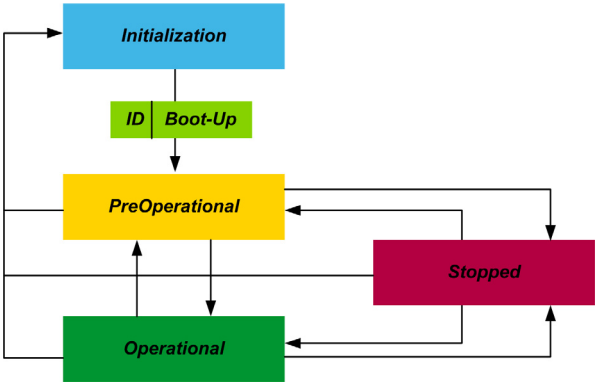
The object dictionary entries of a CANopen device are described in an electronic data sheet (EDS) file. CANopen master and slaves parameters can be adjusted in the SoMachine CANopen configurator. For more information about CANopen configuration, refer the online help CoDeSys part, Editors/Device Editors/CANbus Configuration Editor chapter.

NMT Protocol

Network Management (NMT) Protocol

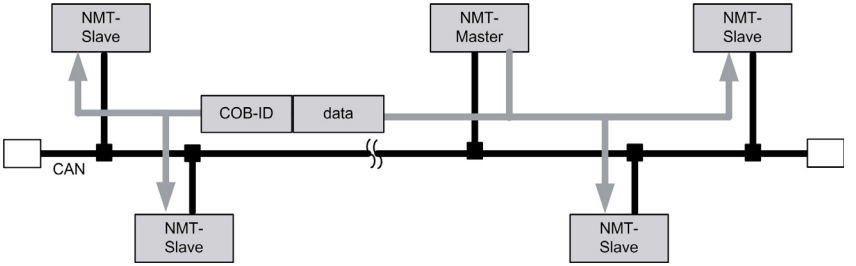
The purpose of the network management (NMT) protocol is to control the network behavior of the CANopen NMT slave devices. Either one dedicated or all network participants are switched through the NMT protocol in their NMT slave state machines.

The following figure represents the CANopen NMT slave state machine.



NMT Services over Master-Slave Relationship

All CANopen devices evaluate the incoming NMT commands (command specifier (CS) code in the NMT message). Only CANopen device with the NMT master capability is able to transmit NMT messages. In a CANopen network, only one NMT master is active.



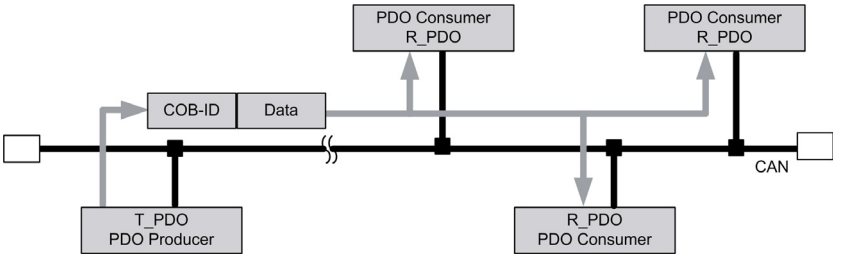
PDO Protocol

Process Data Objects (PDO) Protocol

Process data objects (PDO) are short (8-byte data max), high-prior CAN messages which are transmitted in a broadcast. Therefore, PDO are suitable for the transmission of real-time data such as control and status information of a drive or an I/O module, measured values provided by sensors, etc.

PDO are transmitted in an unconfirmed manner. This means, even after any specific network participant receives the information, there is no acknowledgement of the receipt.

The following figure represents the PDO transmission from producer (Transmit PDO) to consumers (Receive PDO).



Triggering Events for PDO Transmission

The triggering events for PDO transmission are defined below.

Triggering Event	Definition
Event or timer-driven	A device-internal event triggers the PDO transmission (for example, value change, temperature value exceeds a certain limit, event-timer elapsed, etc.)
Remotely requested	PDO transmission requested by remote transmission request (RTR).
Cyclic synchronous	The transmission of the PDO is coupled to the reception of the SYNC message.
Acyclic synchronous	A defined device-specific event triggers the PDO, which transmits with the reception of the next SYNC message.

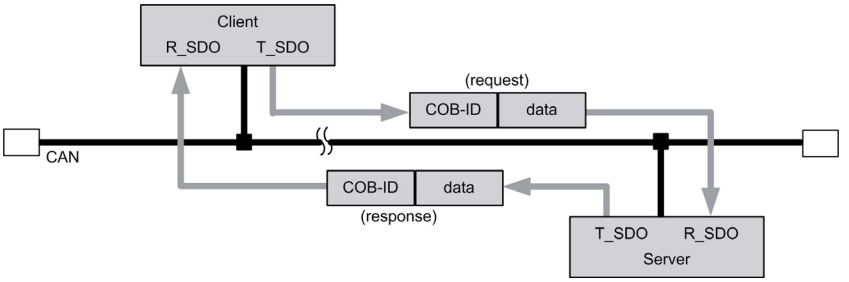
SDO Protocol

Service Data Objects (SDO) Protocol

Service data objects (SDO) allow the access to any entry of the CANopen object dictionary (OD). An SDO establishes a peer-to-peer communication channel between two devices. In addition, the SDO protocol enables to transfer any amount of data in a segmented way. Therefore, the SDO protocol is mainly used in order to communicate with configuration data.

An SDO connection between two devices is established by configuring the related SDO server respectively with the client channel. SDO transmission is a confirmed service.

The following figure represents the SDO exchange with request and response.

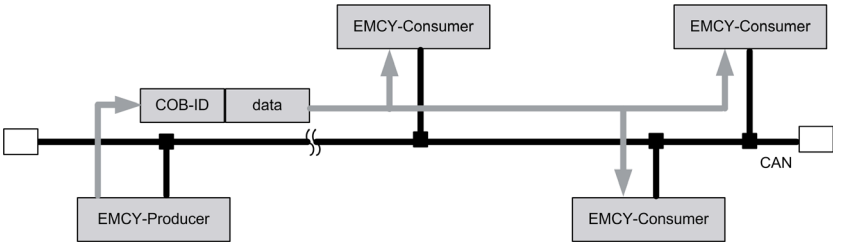


EMCY Protocol

Emergency (EMCY) Protocol

The emergency (EMCY) object enables devices to indicate device-internal detected errors. When they receive this signal, other network participants can evaluate the received information and start appropriate, manufacturer-specific counter actions.

The following figure represents the detected error message through EMCY objects.



EMCY Messages

The EMCY messages received from other CANopen devices are stored in an EMCY storage table. For each EMCY producer, if no EMCY message is received, or as long as the most recent EMCY message was a *no error* message, EMCY consumers consider the EMCY producer to have no internal detected error.

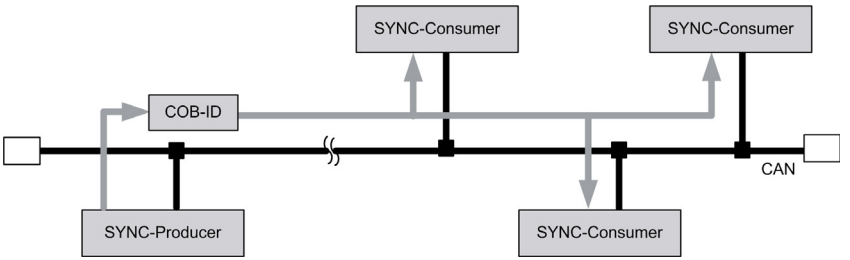
SYNC Protocol

Synchronous (SYNC) Protocol

Synchronous network behavior can be achieved with the SYNC protocol. The cyclically transmitted SYNC message indicates the consumers to start their application-specific behavior, which is coupled to the reception of the SYNC message.

Once data has been received, a Synchronous PDO considers the data to be valid upon receipt of the SYNC message.

The following figure represents the SYNC messages transmission of SYNC protocol.

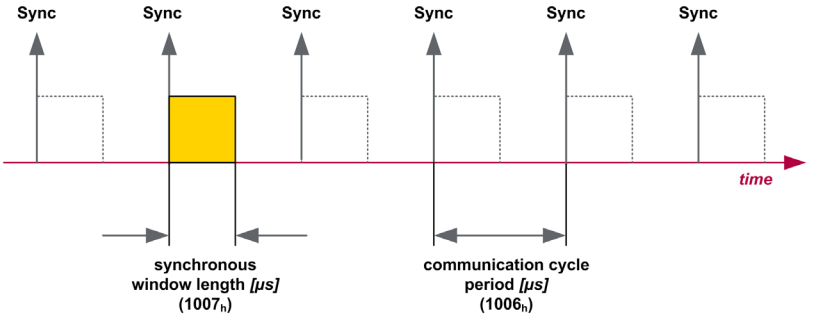


Control Variables for the SYNC behavior

The time period between two consecutive SYNC messages is called *communication cycle period* and can be adjusted in the object dictionary of the SYNC producer (object index 1006 hex).

The synchronous transmit PDO are transmitted within a given time window after the reception of the SYNC message. This time window is called *synchronous window length* and is configurable in the object dictionary of all devices which have to transmit synchronous PDO (object index 1007 hex).

The following figure represents the synchronous window length and communication cycle period.



Error Control Protocol

Error Control Protocol

Heartbeat or guarding services are network management (NMT) services used to check the presence of network participants and to get their state.

Two types of error control protocols exist in CANopen:

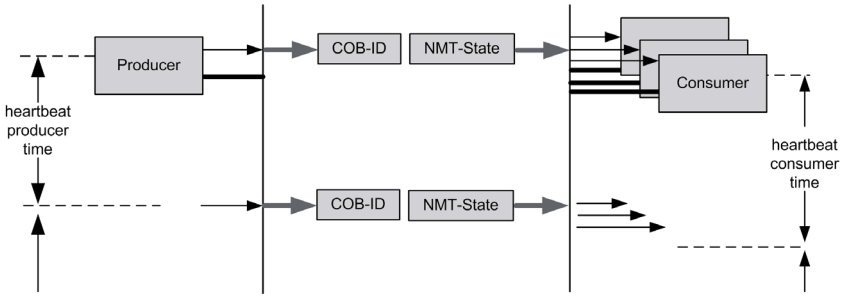
- Heartbeat
- Node/Life Guarding

Because the heartbeat protocol is the more flexible one and works without remote transmission request (RTR), it is you should use it if it is supported by your CANopen devices.

Heartbeat

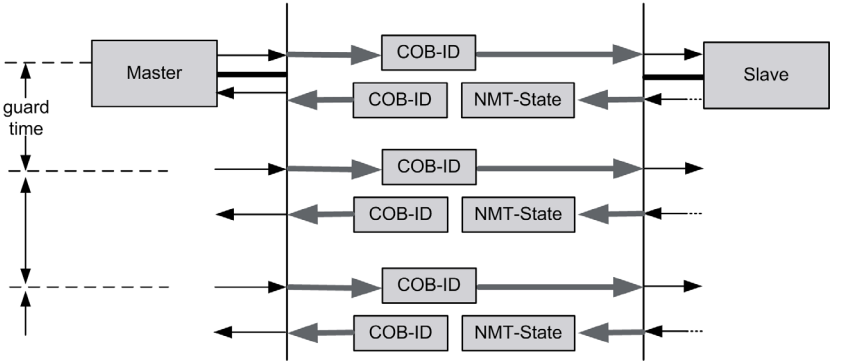
A CANopen device transmits its heartbeat cyclically. The cycle time is called *heartbeat producer time*. On reception of the heartbeat message, the heartbeat consumers (for example, the CANopen manager) evaluate whether the related device is still working properly and remains in the correct NMT-state. The period of time in which at least one heartbeat message has to be received from the related heartbeat producer is configured in the consumers (heartbeat consumer time).

The following figure represents the heartbeat with time intervals of the error control protocol.



Node/Life Guarding

The CANopen manager requests the error control message through a remote transmission request (RTR). The device to be guarded replies with a CAN data frame which indicates the current NMT-state.



Master: The master monitors the slave NMT-state (Node Guarding)
Slave: The slave monitors the master signs of *life* (Life Guarding)

Introduction to CiA405

2

Introduction

Methods of Accessing the CANopen Network

The CANopen interface and device profile for IEC 61131-3 programmable controllers (CiA405) describes two methods of accessing the CANopen network from the controller:

- network variables
- function blocks

Network Variables

The network variables are usually mapped into PDO to be received or to be transmitted. In the object dictionary, the IEC 61131-3 variables are accessible in a defined range of indexes.

Function Blocks

The profile also defines some CANopen-specific function blocks, for example, SDO, NMT and Emergency communication services.

The CAA CiA 405 Library

The CAA CiA 405 Library offers a set of function blocks to meet the requirements of the CiA405 for the access to the CANopen network from the application (IEC61131-3 program) of the controller (CANopen master). The library is automatically declared in the controller library manager when a CANopen manager is added to a controller CAN bus interface.

In the library, the function blocks are organized as follows:

- `Network management` function blocks:
  - `CIA405.NMT`: to control the CANopen devices NMT-states
  - `CIA405.RECV_EMCY`: to scan the EMCY-storages over all devices
  - `CIA405.RECV_EMCY_DEV`: to get the last stored EMCY message of a specified device
- `Own node id` function block:
  - `CIA405.GET_LOCAL_NODE_ID`: to get the controller CANopen manager node ID
- `Query state` function blocks:
  - `CIA405.GET_CANOPEN_KERNEL_STATE`: to get the CANopen kernel current state
  - `CIA405.GET_STATE`: to get a specified device current state
- `SDO access` function blocks:
  - `CIA405.SDO_READ`: to read any-size objects of a specified device
  - `CIA405.SDO_READ4`: to read up to 4-byte objects of a specified device
  - `CIA405.SDO_WRITE`: to write any-size objects of a specified device
  - `CIA405.SDO_WRITE4`: to write up to 4-byte objects of a specified device

NOTE: For the control of Altivar drives and Lexium motion drives on CANopen, prefer the use of dedicated PLCopen function blocks (Refer *Altivar, Integrated Lexium* and *Lexium Library Guides*).

Namespace

The CAA CiA 405 library namespace is `CIA405`. The library function blocks, variables and constants have to be used with the library namespace.

Example of function block instance and variable declaration:

```
VAR
ReadObject: CIA405.SDO_READ;
SDOabort_info: CIA405.SDO_ERROR;
END_VAR
```

Function Blocks Descriptions

II

Overview

This part gives an overview of the various function blocks of CAA CiA 405 Library used to manage and monitor CANopen network and devices from the controller applications.

What's in this Part?

This part contains the following chapters:

Function Blocks Common I/O and Behavior

3

3.1 Common I/O and Behavior

What's in this Section?

This section contains the following topics:

Common I/O Description

Introduction

This topic describes the generic management and executions of the CAA CiA 405 Library function blocks using the `CIA405.RECV_EMCY` function block as an example. The I/O common to all function blocks are described hereafter. They are inherited from the `CiA405Base` internal hidden function block.

Graphical Representation

The parameters that are common to all function blocks in the CAA CiA 405 library are highlighted in the graphic below.

Common Input Variables Description

The following table describes the input parameters common to all function blocks of the CAA CiA 405 Library.

Input	Data Type	Description
`NETWORK`	`USINT`	CAN channel on which the requested service must be executed. 1 (default value) = first CAN bus interface 2 = second CAN bus interface (if exists)
`ENABLE`	`BOOL`	Enables the execution of the function block. On rising edge: the execution starts. On falling edge: the execution is canceled if it is not complete. Otherwise, output data are reset to 0.
`TIMEOUT`	`UDINT`	Maximum execution time in ms. If timeout is reached before a response is available, the execution is aborted on timeout error. 0 (facory setting) = timeout is disabled. 1...65535 = timeout value in ms.

Common Output Variables Description

The following table describes the output variables common to all function blocks of the CAA CiA 405 Library.

Output	Data Type	Description
CONFIRM	BOOL	TRUE when the execution is completed successfully.
ERROR	CANOPEN_KERNEL_ERROR (USINT)	Contains the execution detected error code returned by the CANopen kernel. 00 hex = no execution error detected. 01 hex = detected error, but code available at another function block output (not from CANopen kernel). 02…FF hex = CANopen kernel detected error code.

CANopen Kernel Detected Error Codes

Description

The library offers a CIA405.CANOPEN_KERNEL_ERROR_CODES global variables list with predefined error code variables.

Detected Error Code

The error codes with associated global variable and description are given below.

Detected Error Code	Description
CANOPEN_KERNEL_NO_ERROR = 00 hex	The CANopen kernel has not detected an error.
CANOPEN_KERNEL_OTHER_ERROR = 01 hex	If ERROR output = 01 hex, an error has been detected and, if there is another error code output at the function block, this output contains a more specific information. **Examples:** • CIA405.SDO_READ and CIA405.SDO_WRITE function blocks: the ERRORINFO output contains the content of the SDO-Abort message. • CIA405.RECV_EMCY and CIA405.RECV_EMCY_DEV function blocks: the ERRORINFO output contains the content of the received EMCY message. When an emergency message is available at ERRORINFO output, the ERROR output =1 and CONFIRM output = 0. • CIA405.GET_CANOPEN_KERNEL_STATE function block: 01 hex value is never delivered as there is no other error code output.
CANOPEN_KERNEL_DATA_OVERFLOW = 02 hex	Overflow on send-buffers or receive-buffers of CANopen objects.
CANOPEN_KERNEL_TIMEOUT = 03 hex	A function block execution timeout occurred.
CANOPEN_KERNEL_CANBUS_OFF = 10 hex	The CANopen node is disconnected from the CAN bus.
CANOPEN_KERNEL_CAN_ERROR_PASSIVE = 11 hex	The CANopen node is error passive: the node is able to communicate, but not allowed to send an active error flag in case of error detection.
CANOPEN_INTERNAL_FB_ERROR = 21 hex	Manufacturer specific error code. **NOTE:** 21hex to FFhex are device manufacturer dedicated values.

Function Block Execution Diagrams

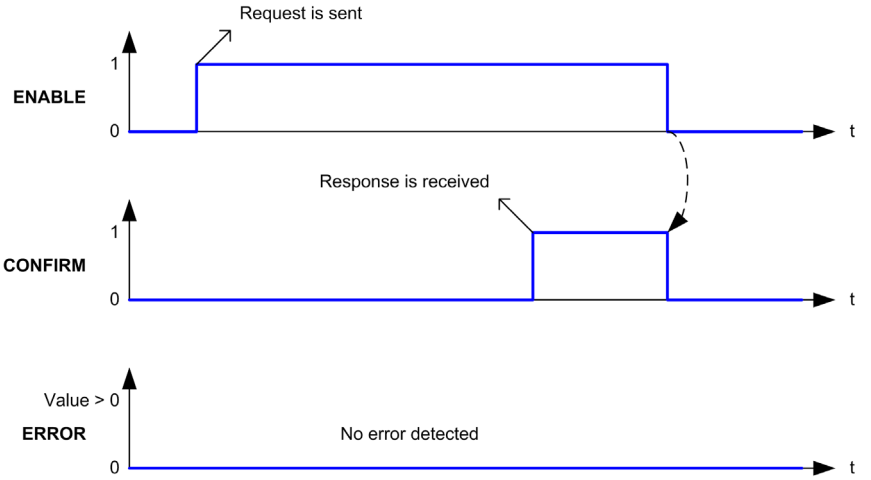
Behavior of the Control Signals

The three typical behaviors of the control signals ENABLE, CONFIRM and ERROR are described below:

- Execution ended without detected error
- Execution cancelled by application
- Execution aborted or ended on detected error

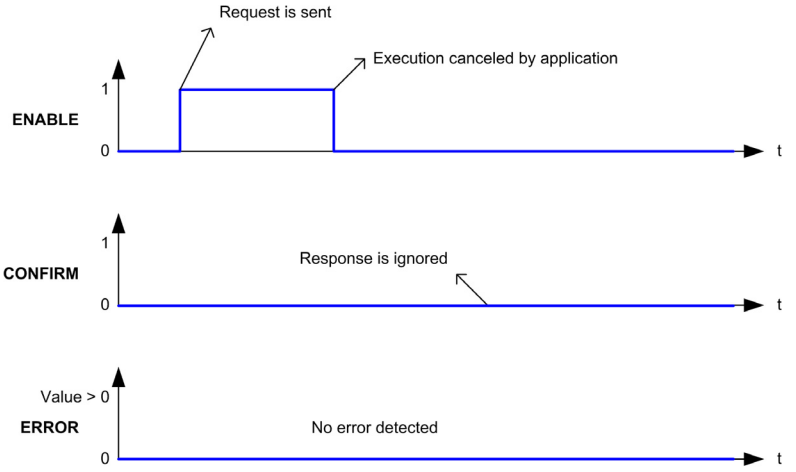
Execution Ended without Detected Error

As soon as the response to the current request is availabe and no error has been detected, the output CONFIRM is set to TRUE and stays TRUE as long as the function block is called with input ENABLE is TRUE. When the function block is called with ENABLE reset to FALSE, CONFIRM is reset to FALSE and the function block is ready for a new execution start.



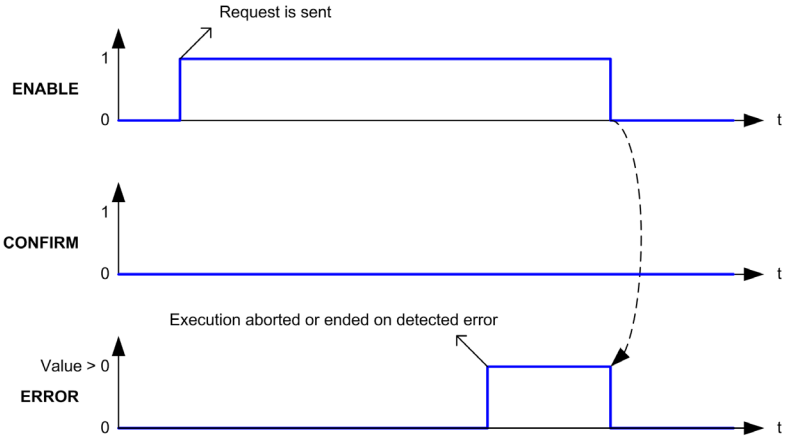
Execution Cancelled by Application

If the function block is called with ENABLE reset to FALSE before the current execution has ended, the function block execution is cancelled. A possibly available or later arriving response to the cancelled request is ignored.



Execution Aborted or Ended on Detected Error

As soon as the current execution has been aborted by an SDO abort message received or has ended on a detected error, the output ERROR is set to a value different from 0 (refer CANopen Kernel Detected Errors Codes *(see page 28)* for further information about detected error codes). The output ERROR is reset to 0 and the function block is ready for a new execution start when the function block is called with input ENABLE reset to FALSE.



Network Management Function Blocks

4

What's in this Chapter?

This chapter contains the following sections:

4.1 CIA405.NMT: Devices NMT-State Management

Devices NMT-state Management

Function Block Description

The CIA405.NMT function block allows the control of the NMT-state of the CANopen devices from the controller application. It performs NMT service request to a CANopen targeted device to perform the requested NMT-state transition.

Graphical Representation

Specific Input Variables Description

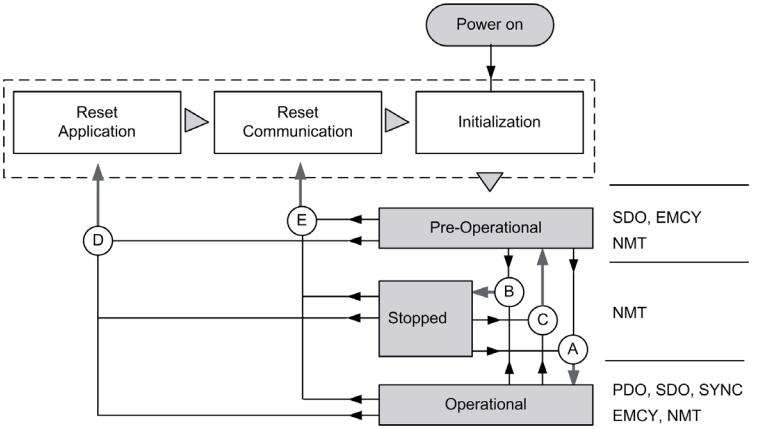
Input	Data Type	Description
DEVICE	DEVICE(USINT)	CANopen targeted device node ID. 0 (default value) = all NMT slave devices 1...127 = targeted device node ID
STATE	TRANSITION_STATE. Refer below for details	Requested NMT-state transition.

NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

CIA405.TRANSITION_STATE ENUM

The NMT-state machine describes the initialising and state of an NMT slave in main operation.

The figure below shows NMT-states, associated available communication objects (PDO, SDO, SYNC, EMCY and NMT) and 5 state transitions (A to E).

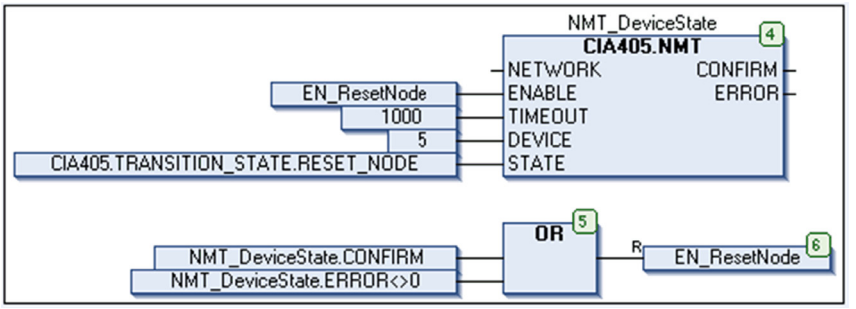


The CIA405.TRANSITION_STATE enumerated type contains the NMT-state transition commands which are described in the below table.

Enumerator	Value (hex)	Description
STOP_REMOTE_NODE	0004	Switch to *Stopped* state. (transition B)
START_REMOTE_NODE	0005	Switch to *Operational* state. (transition A)
RESET_NODE	0006	Switch to *Reset Application* state. Load saved data of the device profiles and switch automatically to *Pre-Operational* state through *Reset Communication.* (transition D)
RESET_COMMUNICATION	0007	Switch to *Reset Communication* state. Load stored data of the communication profile and switch automatically to *Pre-Operational* state. (transition E)
ENTER_PRE_OPERATIONAL	007F	Switch to *Pre-Operational* state. (transition C)
ALL_EXCEPT_NMT_AND_SENDER	0800	Not implemented (invalid parameter)

Example

The example below shows how to send a Reset Node command to CANopen node 5 connected to the first CAN bus interface with a 1s (1000ms) timeout. The command is sent when the boolean variable `EN_ResetNode` is set to TRUE (by user online or application). `EN_ResetNode` command is reset to FALSE when the execution has ended successfully (output `CONFIRM` = TRUE) or an error is detected (output `ERROR` <> 0).



4.2 CIA405.RECV_EMCY: EMCY Messages Scanning

EMCY Messages Scanning

Function Block Description

The CIA405.RECV_EMCY function block scans the emergency (EMCY) message storages in a loop over all existing CANopen devices and returns EMCY messages found.

When the function block is enabled, the EMCY storages scan is launched from the previous scan stop point:

- If an EMCY message different from *No Error* is found, the scan ends at this point and the function block returns the EMCY message and associated device node ID.
- If a *No Error* EMCY message is found which was previously different from *No Error*, the scan ends at this point and the function block returns an EMCY message = 0 and associated device node ID.
- If no EMCY message is different from *No Error* and no new *No Error* EMCY message is found during one complete scan, the function block returns EMCY message = 0 and device node ID = 0.

Graphical Representation

Specific Output Variables Description

Output	Data Type	Description
DEVICE	DEVICE (USINT)	Node ID of the CANopen device associated to the returned EMCY message. 0 = No EMCY message found 1...127 = device node ID
ERRORINFO	CS.EMCY_ERROR. Refer CS.EMCY_ERROR structure *(see page 38)* for details	Last EMCY message received from the CANopen device with node ID *DEVICE*.

NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

4.3 CIA405.RECV_EMCY_DEV: Get Device EMCY Message

Get Device EMCY Message

Function Block Description

The CIA405.RECV_EMCY_DEV function block returns the last stored emergency (EMCY) message received from a specified CANopen device.

Graphical Representation

Specific Input Variables Description

Input	Type	Description
DEVICE	DEVICE(USINT)	Node ID of the CANopen device to be checked. 0 (default value) = local device (controller) 1…127 = device node ID

Specific Output Variables Description

Output	Type	Description
ERRORINFO	CS.EMCY_ERROR. (Refer below for details)	Last EMCY message received from the CANopen device with node ID *DEVICE*.

NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

CS.EMCY_ERROR Structure

The CS.EMCY_ERROR is the structure associated to the EMCY message content (Refer to your device CANopen guide for further information on EMCY message specific content). The CS.EMCY_ERROR contains the following elements.

Element	Type	Description
EMCY_ERROR_CODE	WORD	Error code of the EMCY message
ERROR_REGISTER	BYTE	Error register (bit field) of the EMCY message
ERROR_FIELD	ARRAY [1...5]OF BYTE	Device manufacturer specific error field of the EMCY message

NOTE: This structure is declared in the CAA CANopen Stack library (namespace = CS). This is why the structure full name is used (<namespace>.<data type>). The CAA CiA 405 library namespace is CIA405.

Own Node ID Function Blocks

5

5.1 CIA405.GET_LOCAL_NODE_ID: Get Controller CANopen Node ID

Get Controller CANopen Node ID

Function Block Description

The CIA405.GET_LOCAL_NODE_ID function block returns the controller CANopen node ID on specified CAN bus interface.

Graphical Representation

Specific Output Variables Description

Output	Type	Description
DEVICE	DEVICE (USINT)	CANopen node ID of the controller. 0 = not valid 1...127 = controller node ID

NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

Query State Function Blocks

6

What's in this Chapter?

This chapter contains the following sections:

6.1 CIA405.GET_CANOPEN_KERNEL_STATE: Get CANopen Kernel State

Get CANopen Kernel State

Function Block Description

The CIA405.GET_CANOPEN_KERNEL_STATE function block returns the current state of the controller CANopen kernel.

Graphical Representation

Specific Output Variables Description

Output	Type	Description
STATE	CANOPEN_KERNEL_ERROR (Refer CANopen Kernel Detected Error Codes *(see page 28)* for details)	Current state of the controller CANopen kernel
DEVICE	DEVICE (USINT)	CANopen node ID of the controller. 0 = invalid 1...127 = controller node ID

NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

6.2 CIA405.GET_STATE: Get CANopen Device State

Get CANopen Device State

Function Block Description

The CIA405.GET_STATE function block returns the current NMT-state of the specified CANopen device if heartbeat or node guarding is active.

Graphical Representation

Specific Input Variables Description

Input	Type	Description
DEVICE	DEVICE (USINT)	Node ID of the CANopen device to be checked. 0 (default value) = local device (controller) 1...127 = device node ID

Specific Output Variables Description

Output	Type	Description
STATE	DEVICE_STATE (Refer below for details)	NMT-state of the CANopen device.

NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

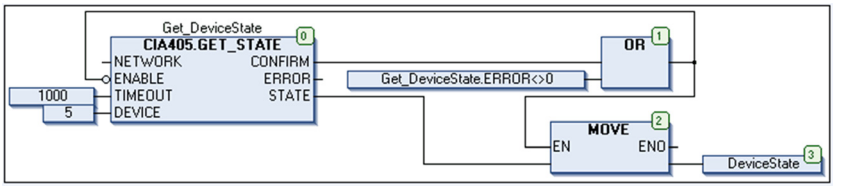
CIA405.DEVICE_STATE ENUM

The CIA405.DEVICE_STATE enumerated type contains the list of CANopen device NMT-states.

Enumerator	Value	Description
INIT	0	*Initialization state*
RESET_COMM	1	*Reset Communication state*
RESET_APP	2	*Reset Application state*
PRE_OPERATIONAL	3	*Pre-Operational state*
STOPPED	4	*Stopped state*
OPERATIONAL	5	*Operational state*
UNKNOWN	6	Unknown NMT-state. The node guarding or heartbeat is not active for the selected device or the controller is not the heartbeat consumer.
NOT_AVAIL	7	NMT-state not available. The node guarding or heartbeat is active for the selected device but the device does not report its NMT-state correctly before timeout.

Example

The example below shows how to get the state of the CANopen node 5 connected to the first CAN bus interface with a 1s (1000ms) timeout. The CIA405.GET_STATE function is automatically executed for a continuous state reading. The device NMT-state is copied into the variable DeviceState of type CIA405.DEVICE_STATE.



SDO Access Function Blocks

7

What's in this Chapter?

This chapter contains the following sections:

7.1 CIA405.SDO_READ: Read Any Size CANopen Objects

Read Any Size CANopen Objects

Function Block Description

The CIA405.SDO_READ function block is used to read an any size CANopen object of a specified device through SDO messages.

These specific parameters must be passed to the function block:

- device node ID
- SDO client/server channel (by default, only one channel is defined)
- CANopen object index/sub-index
- pointer to the data buffer in which object values will be stored
- data buffer size

The function block returns the read object size if the reading has ended with success. The data are available in the data buffer.

NOTE: If the size of the object to be read is less than or equal to 4 bytes, prefer using the CIA405.SDO_READ4 function block.

Graphical Representation

Specific Input Variables Description

Input	Type	Description
DEVICE	DEVICE (USINT)	Node ID of the CANopen device. 0 (default value) = local device (controller) 1...127 = device node ID
CHANNEL	USINT	SDO channel number. Default value = 1
INDEX	WORD	Object index. Range: 0000 hex … FFFF hex
SUBINDEX	BYTE	Object sub-index. Range: 00 hex … FF hex
DATA	POINTER TO BYTE	Address of the data buffer that receives the data read from the device object. The ADR standard function must be used to define the associated pointer.

Specific Output Variables Description

Output	Type	Description
ERRORINFO	SDO_ERROR (UDINT)	When ERROR output = 1, returns the SDO-abort message content (4-byte size).

Specific Input/Output Variables Description

Input/Output	Type	Description
DATALENGTH	UINT	As an input: size of the data buffer (in bytes). **NOTE:** To be sure the DATALENGTH input is correctly initialized (at function block execution starts on ENABLE input rising edge) with the data buffer size, use the SIZEOF standard function. As an output: size of the object read (in bytes).

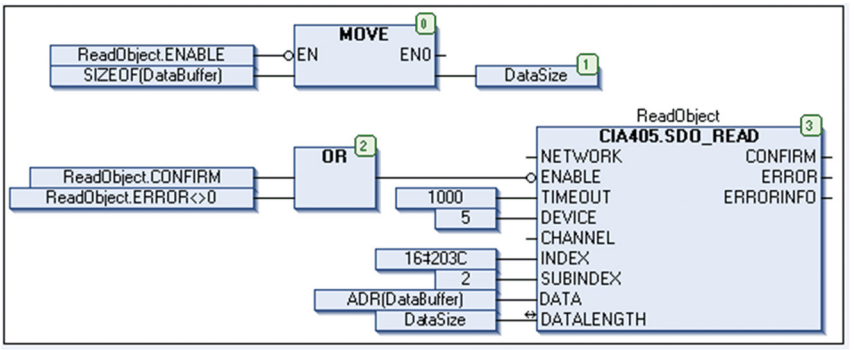
NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

Example

The example below shows how to read the object index 203C hex/sub-index 02 hex of the CANopen node 5 connected to the first CAN bus interface with a 1s (1000ms) timeout. The `CIA405.SDO_READ` function block instance (ReadObject) is automatically executed for a continuous reading.

The variable `DataSize` (`UINT` type):

- is initialized with the size of the buffer of data (DataBuffer: array of N bytes) when `ENABLE` input of the function block is FALSE, before launching the next execution.
- contains the size of data read (in bytes) at rising edge of the function block output `CONFIRM` (the example neither shows how to extract the value from the buffer of data nor shows how to manage error detections).



7.2 CIA405.SDO_READ4: Read upto 4-Byte CANopen Objects

Read upto 4-Byte CANopen Objects

Function Block Description

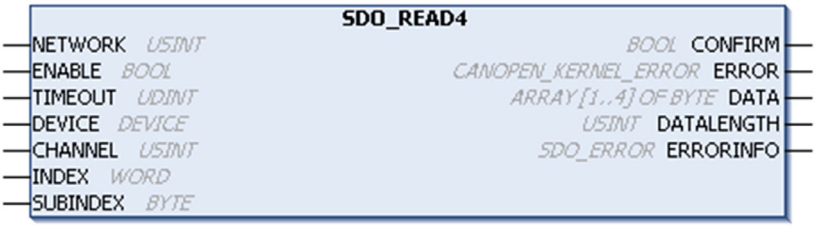
The CIA405.SDO_READ4 function block is used to read an up to 4-byte size CANopen object of a specified device through a SDO message.

These specific parameters must be passed to the function block:

- Device node ID
- SDO client/server channel (by default, only one channel is defined)
- CANopen object index/sub-index

The function block returns the read object size if the reading was completed successfully. The data are available in a 4-byte array.

Graphical Representation



Specific Input Variables Description

Input	Type	Description
DEVICE	DEVICE (USINT)	Node ID of the CANopen device. 0 (default value) = local device (controller) 1...127 = device node ID
CHANNEL	USINT	SDO channel number. Default value = 1
INDEX	WORD	Object index. Range: 0000 hex … FFFF hex
SUBINDEX	BYTE	Object sub-index. Range: 00 hex … FF hex

Specific Output Variables Description

Output	Type	Description
DATA	ARRAY[1...4] OF BYTE	Data array that receives the data read from the device object.
DATALENGTH	USINT	Size of the read object (in bytes).
ERRORINFO	SDO_ERROR (UDINT)	When ERROR output = 1, returns the SDO-abort message content (4-byte size).

NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

The following table shows the object size versus DATA array content.

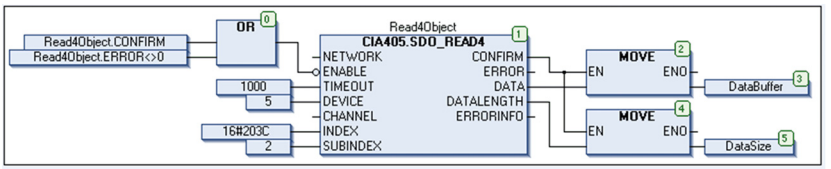
Object size Example	DATALENGTH	DATA(1)	DATA(2)	DATA(3)	DATA(4)
1 byte **01 hex**	1	**01 hex**	not significant	not significant	not significant
2 bytes **01 23 hex**	2	LSB **23 hex**	MSB **01 hex**	not significant	not significant
3 bytes **01 23 45 hex**	3	LSB **45 hex**	**23 hex**	MSB **01 hex**	not significant
4 bytes **01 23 45 67 hex**	4	LSB **67 hex**	**45 hex**	**23 hex**	MSB **01 hex**

LSB = low significant byte

MSB = most significant byte

Example

The example below shows how to read the object index 203C hex/sub-index 02 hex of the CANopen node 5 connected to the first CAN bus interface with a 1s (1000ms) timeout. The CIA405.SDO_READ4 function block instance (Read4Object) is automatically executed for a continuous reading. The variable DataBuffer (array of 4 bytes) contains the value of the last data read. The variable DataSize (USINT type) contains the size (maximum 4 bytes) of the last data read. The example does not show how to manage error detections.



7.3 CIA405.SDO_WRITE: Write Any Size CANopen Objects

Write Any Size CANopen Objects

Function Block Description

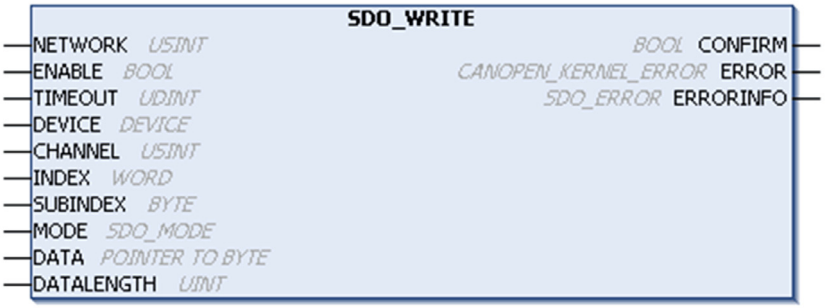
The CIA405.SDO_WRITE function block is used to write an any size CANopen object of a specified device through SDO messages.

These specific parameters must be passed to the function block:

- device node ID
- SDO client/server channel (by default, only one channel is defined)
- CANopen object index/sub-index
- SDO mode (defines the data transmission mode, refer SDO_MODE ENUM for details *(see page 53)*
- pointer to the data buffer in which object values to be written are stored
- number of bytes to write

NOTE: If the size of the object to write is less than or equal to 4 bytes, use the CIA405.SDO_WRITE4 function block.

Graphical Representation



Specific Input Variables Description

Input	Type	Description
DEVICE	DEVICE (USINT)	Node ID of the CANopen device. 0 (default value) = local device (controller) 1...127 = device node ID
CHANNEL	USINT	SDO channel number. Default value = 1
INDEX	WORD	Object index. Range: 0000 hex … FFFF hex
SUBINDEX	BYTE	Object sub-index. Range: 00 hex … FF hex
MODE	SDO_MODE (Refer below for details)	Data transmission mode. 0 (default value) = AUTO for automatic mode selection
DATA	POINTER TO BYTE	Address of the data buffer in which object values to write are stored. The ADR standard function must be used to define the associated pointer.
DATALENGTH	UINT	Size of the object to write (in bytes). **NOTE:** Use the SIZEOF standard function to define the data buffer size.

Specific Output Variables Description

Output	Type	Description
ERRORINFO	SDO_ERROR (UDINT)	When ERROR output = 1, returns the SDO-abort message content (4-byte size).

NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

CIA405.SDO_MODE ENUM

The CIA405.SDO_MODE enumerated type contains the list of SDO transmission modes.

Enumerator	Value	Description
AUTO	0	Automatic mode selection.
EXPEDITED	1	SDO Expedited mode for up to 4-byte data. Data sent in one SDO request.
SEGMENTED	2	SDO Segmented mode for more than 4-byte data. Data are segmented in 7-byte segments sent by consecutive SDO acknowledged requests.
BLOCK	3	SDO Block mode for more than 4-byte data. Data sent by consecutive frames are segmented into 7-byte blocks, that are not acknowledged. An acknowledgement is sent by the receiver once all the blocks have been received. **NOTE:** This is a faster transmission mode that may not be supported by your device because this mode is a recent CANopen profile addition.

Example

Program example in ST language to Read & Write data in CANopen devices:

One ATV71 is declared at Node address 1

One ATV31 is declared at Node address 2

Data to write or to read is:

INDEX:=16#203C,

SUBINDEX:=2,

The written value (SDOWrite_data) is a 0 to 500 ramp incremented at each cycle.

```
PROGRAM Read_Write_SDO
VAR
   WriteSDO_ATV71 : CIA405.SDO_WRITE;
   WriteSDO_ATV31 : CIA405.SDO_WRITE;
   ReadSDO_ATV31 : CIA405.SDO_READ;
   SDOWrite_data : UINT;
   SDORead_data : UINT;
   ReadSDO_datalength : UINT:=2;
   start_test : BOOL;
END_VAR
```

```
IF start_test THEN
   WriteSDO_ATV31(
      NETWORK:= ,
      ENABLE:= NOT (WriteSDO_ATV31.CONFIRM OR
   (WriteSDO_ATV31.ERROR<>0)) ,
      TIMEOUT:= ,
      DEVICE:=2 ,
      CHANNEL:= ,
      INDEX:=16#203C ,
      SUBINDEX:=2 ,
      MODE:= ,
      DATA:=ADR(SDOWrite_data) ,
      DATALENGTH:=2 ,
      CONFIRM=> ,
      ERROR=> ,
      ERRORINFO=> );
   ReadSDO_ATV31(
      NETWORK:= ,
      ENABLE:=NOT (ReadSDO_ATV31.CONFIRM OR
   (ReadSDO_ATV31.ERROR<>0)) ,
      TIMEOUT:= ,
      DEVICE:=1 ,
      CHANNEL:= ,
      INDEX:=16#203C ,
      SUBINDEX:=2 ,
      DATA:=ADR(SDORead_data) ,
      DATALENGTH:=ReadSDO_datalength ,
      CONFIRM=> ,
      ERROR=> ,
      ERRORINFO=> );
   WriteSDO_ATV71(
      NETWORK:= ,
      ENABLE:= NOT (WriteSDO_ATV71.CONFIRM OR
```

```
   (WriteSDO_ATV71.ERROR<>0)) ,
      TIMEOUT:= ,
      DEVICE:=1 ,
      CHANNEL:= ,
      INDEX:=16#203C ,
      SUBINDEX:=2 ,
      MODE:= ,
      DATA:=ADR(SDOWrite_data) ,
      DATALENGTH:=2 ,
      CONFIRM=> ,
      ERROR=> ,
      ERRORINFO=> );
   IF SDOWrite_data<500
      THEN SDOWrite_data:=SDOWrite_data+1;
      ELSE SDOWrite_data:=0;
   END_IF
END_IF
```

7.4 CIA405.SDO_WRITE4: Write upto 4-Byte CANopen Objects

Write upto 4-Byte CANopen Objects

Function Block Description

The CIA405.SDO_WRITE4 function block is used to write an up to 4-byte size CANopen object of a specified device through SDO messages.

These specific parameters must be passed to the function block:

- device node ID
- SDO client/server channel (by default, only one channel is defined)
- CANopen object index/sub-index
- values to write
- number of bytes to write (object size)

Graphical Representation

Specific Input Variables Description

Input	Type	Description
DEVICE	DEVICE (USINT)	Node ID of the CANopen device. 0 (default value) = local device (controller) 1...127 = device node ID
CHANNEL	USINT	SDO channel number. Default value = 1
INDEX	WORD	Object index. Range: 0000 hex … FFFF hex
SUBINDEX	BYTE	Object sub-index. Range: 00 hex … FF hex
DATA	ARRAY[1...4] OF BYTE	Data array in which object values to write are stored.
DATALENGTH	USINT	Size of the object to write (in bytes).

Specific Output Variables Description

Output	Type	Description
ERRORINFO	SDO_ERROR (UDINT)	When ERROR output = 1, returns the SDO-abort message content (4-byte size).

NOTE: For function block common I/O descriptions and execution behavior, refer to the chapter Function Blocks Common I/O and Behavior *(see page 25)*.

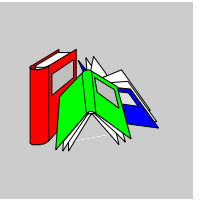
The following table shows the object size versus DATA array content.

Object size Example	DATALENGTH	DATA(1)	DATA(2)	DATA(3)	DATA(4)
1 byte **01 hex**	1	**01 hex**	not significant	not significant	not significant
2 bytes **01 23 hex**	2	LSB **23 hex**	MSB **01 hex**	not significant	not significant
3 bytes **01 23 45 hex**	3	LSB **45 hex**	**23 hex**	MSB **01 hex**	not significant
4 bytes **01 23 45 67 hex**	4	LSB **67 hex**	**45 hex**	**23 hex**	MSB **01 hex**

LSB = low significant byte

MSB = most significant byte

Glossary



C

CAN

Controller Area Network (CAN) is a serial bus system for embedded control.

CANopen

CANopen is the internationally standardized (EN 50325-4) CAN-based higher-layer protocol for embedded control system.

CiA

CAN in Automation (CiA) is the group of international users and manufacturers that develops and supports CANopen and other CAN-based higher-layer protocols.

CiA405

The CANopen interface and device profile for IEC 61131-3 programmable controllers

COB

CANopen protocol communication objects

E

EDS

Electronic Data Sheet file for fieldbus device description.

EMCY

Emergency

H

hex

hexadecimal

I

I/O

Input/Output

ID

Identifier

M

ms

millisecond

N

NMT

CANopen network management

O

OD

CANopen protocol object dictionary

P

PDO

CANopen protocol process data objects

Protocol

A set of rules which is used by devices to communicate with each other across a network

S

SDO

CANopen protocol service data objects

Index

N

O

Q

R

S

T