CANopenIA

Serial Remote Access to CANopen for CANgineBerry

for firmware version 1.3 or higher

Rev. 1.15 of 24th April 2019

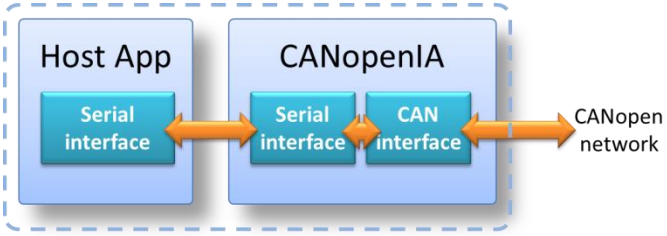
Published by
Embedded Systems Academy GmbH
Bahnhofstraße 17
D-30890 Barsinghausen, Germany
www.esacademy.com

CANgine products by
Embedded Systems Solutions GmbH
Industriestraße 15
D-76829 Landau, Germany
www.essolutions.de

1 Contents

2 Introduction

The CANopen coprocessor (447izer if in CiA447 mode) implements a CANopen device, or a simplified CANopen manager, depending on firmware version. A host system can communicate with the CANopenIA coprocessor via a regular serial channel. The protocol used is ESAcademy's CANopen remote access protocol described in this document. The CANopenIA coprocessor handles all CANopen communication.

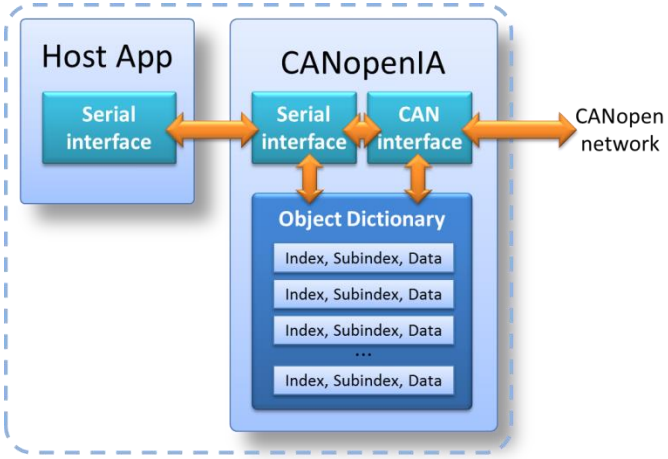


COPROCESSOR WITH HARDWIRED SERIAL INTERFACE

2.1 CANopen Object Dictionary

As required by any CANopen device, the CANopenIA/447izer implements a CANopen Object Dictionary (OD) that contains all configurations of the chip itself as well as all the process data communicated. This OD is available to the CANopen network as well as to the host. Which OD entries are present in the CANopenIA depends on its configuration.

Default configurations are provided for all CANopenIA implementations. Customized configuration file can be generated using the CANopen Architect EDS Editor and transferred into the flash memory of the CANopenIA.



OBJECT DICTIONARY IS ACCESS FROM SERIAL AND CANOPEN SIDE

2.2 CANopen Manager

In CANopen, managers provide several functionalities. The ones provided by CANopenIA are listed in this section.

SDO Client

The CANopenIA-MGR and 447izer versions also support SDO client services. Once such a CANopenIA device is up and running (CANopen state operational), it may send CANopen SDO (Service Data Object) read and write requests to the nodes connected to the CANopen network. This gives the host application read and write access to all the Object Dictionaries of all connected nodes.

Note that in regular CANopen this means that this device uses the regular SDO client channels used by a CANopen Manager. DO NOT use this mode, when another CANopen Manager is present and using these channels at the same time.

NMT Master and Heartbeat monitoring

The CANopenIA-MGR firmware also provides the CANopen NMT (Network Management) Master functions to control the individual nodes connected. The firmware can autostart known devices to facilitate a quick start up of CANopen systems.

A default heartbeat time and timeout monitoring can be automatically activated. If devices are lost (no more heartbeat received), the master automatically transmits a reset request to these nodes for automated recovery support.

Automated PDO handling

The minimal CANopen manager supports an automated PDO (Process Data Object) handling. PDO configurations of connected devices are analyzed and activated. The host is informed about every PDO received from all devices. The information passed on to the host for data received includes node ID, object info (index, subindex) and the data.

2.3 Low level access

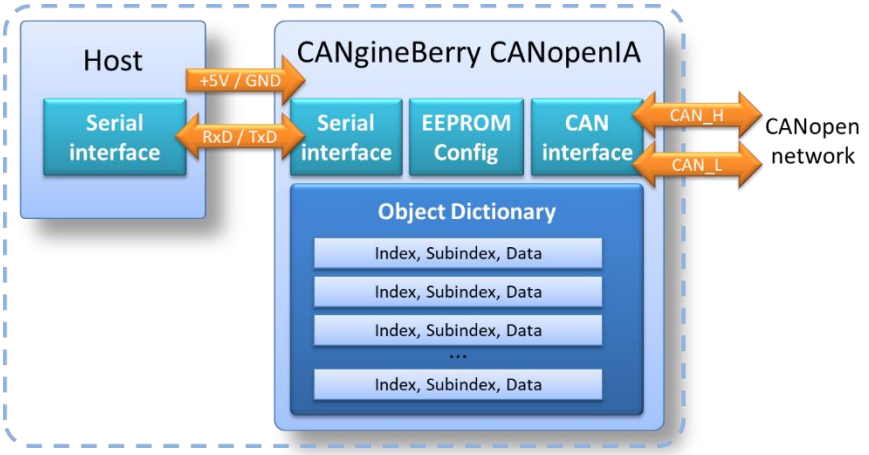
The Software also implements a generic low-level access mode. If this mode is activated, then any CAN message can be transmitted by the host and CAN messages received are reported back to the host. An optional CAN message ID filter allows selecting the CAN messages that should be received.

3 Hardware options

3.1 CANgineBerry

The CANgineBerry is a CANopenIA module intended for the Raspberry Pi. It has a CAN (DB9) connector and only uses 4 pins for the connection to the Raspberry Pi: +5V, GND and the 3.3V Rx/Tx signals of the serial channel. Both CANopen RUN and ERR LEDs are provided.

The module can also be used with any other host system that provides the 4 required pins. If the preferred communication channel to a host is USB, then a USB-UART chip can be used as an interface between the host and the CANgineBerry. In that case verify that the correct voltage levels are used, the CANgineBerry requires +5V for the power supply but only uses 3.3V on the Tx/Rx lines.



CANGINEBERRY SYSTEM OVERVIEW

3.2 CANgine Light

The CANgine Light by Embedded Systems Solutions GmbH is a small device with a CAN connector on one side and a RS232 (DB9) connector on the other. Power is supplied via the CAN connector. The RS232 side can be directly connected to most USB-RS232 converters. Both CANopen RUN and ERR LEDs are provided but no further optional inputs or outputs.

3.3 CANgineII BT

The CANgineII BT by Embedded Systems Solutions GmbH is a small device with a CAN connector and an internal Bluetooth module. Power is supplied via the CAN connector. On the connecting Bluetooth device, the CANgineII BT appears like a generic serial device. Both CANopen RUN and ERR LEDs are provided but no further optional inputs or outputs.

3.4 PCAN-RS232

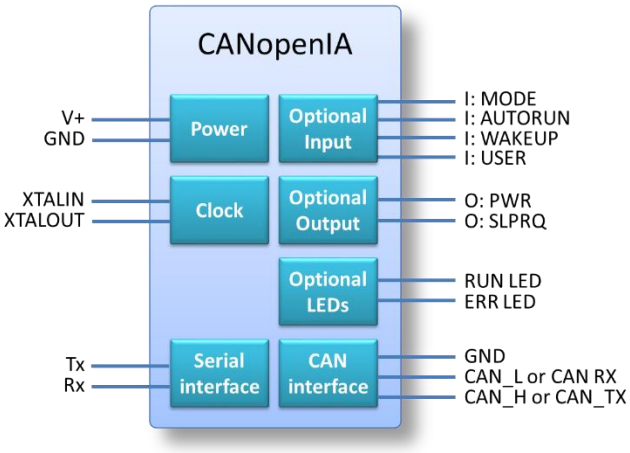
The PCAN-RS232 module is a boxed CAN to RS232 module with industrial style open connectors.

3.5 Library for PCAN Interfaces

The CANopenIA functionality is also available as a Microsoft Windows® DLL for PEAK's CAN_API4 supporting the PCAN interfaces from PEAK System. When used as a library, all commands, responses and indications are provided as functions and call-back functions.

3.6 Custom Module or Chip

The CANopenIA Coprocessor is available as a module or chip for direct integration into your hardware. The number of pins used is minimal, the input, output and LEDs are optional.



SIGNALS OF THE CANOPENIA CHIP OR MODULE

CANopenIA coprocessor implementations are available for the following microcontrollers:

- NXP LPC11C24
  uses internal transceiver, can be directly connected to the CAN_L and CAN_H lines of the CANopen network
- ST-Microelectronics STM32F091 or STM32F042
  requires an external transceiver and connects to the CAN_RX and CAN_TX pins

The optional input signals are:

- MODE:
  Set high if used in CiA 447 mode, else low for regular CANopen mode
- AUTORUN:
  Set high if device should autostart (directly switch itself into operational mode), not recommended for CiA 447
- WAKEUP:
  A rising flank on this pin wake ups the chip / module if it was in sleep, the device then produces the wakeup messages
- USER:
  Input pin for customizations

The optional output pins are

- PWR:
  Produce a rising flank on wakeup
- SLPRQ:
  Set high, if sleep request was received from power manager

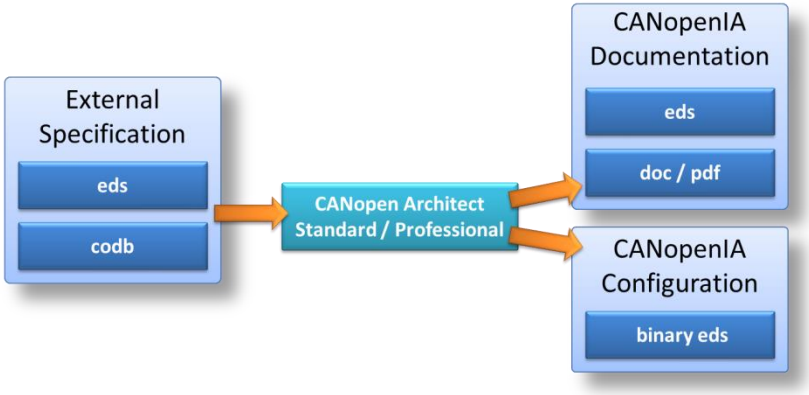
3.7 Wakeup and Sleep

In CiA 447 mode the chip wakes up upon detecting any activity on the serial channel, the CANopen channel or the wakeup pin. It then participates in the CANopen wakeup communication.

Upon reception of the sleep request from the CANopen power manager the device forwards this request to the serial interface and then sets itself into sleep mode.

4 System configuration

The CANopenIA implementation is configured by the local Object Dictionary. This is stored in Flash memory and can be re-loaded. The format used is ESAcademy's binary EDS file format, which is exported by the CANopen Architect EDS utility for Editing CANopen Electronic Data Sheets (EDS). Depending on CANopenIA firmware, a variety of default configuration files are provided.



GENERATING A CONFIGURATION WITH CANOPEN ARCHITECT

In addition, some firmware versions support customized configurations. If an EDS or CODB (CiA format for Object Dictionary definition) already exists specifying the configuration, then this can be imported into the CANopen Architect software. After editing/modifying the configuration, a current EDS and binary EDS are exported. The binary EDS is directly loaded into the CANopenIA device, module or chip.

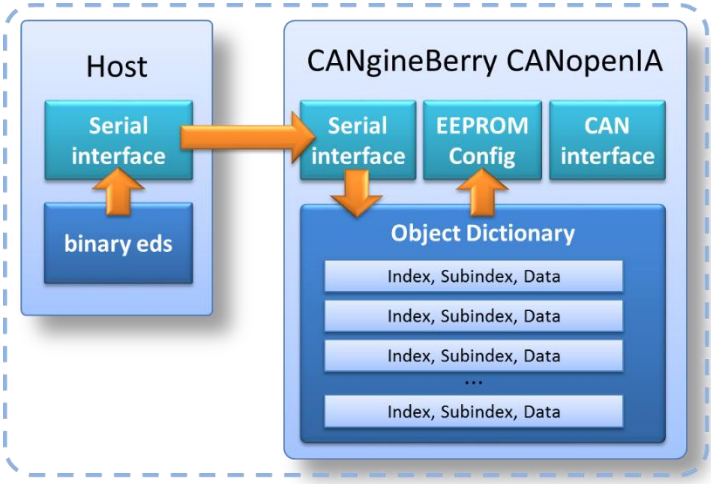
4.1 Bitrate and node ID selection

The bitrate and node ID settings are also made through the binary EDS. The host system cannot change these settings. If the configured node ID is zero, then LSS (Layer Setting Services) are used to get a node ID assigned by the LSS Master.

4.2 Loading a binary EDS

When using a CANopenIA library, the binary EDS file with the configuration to be used is passed to the library upon its initialization. All other implementations store the configuration in EEPROM.

The CANgineBerry supports loading the configuration file through the serial interface (using the provided COIAUpdater utility). This gives the host full configuration control, as it can activate any desired configuration by itself.



LOADING A CANGINEBERRY CONFIGURATION – VIA HOST

4.3 Step-by-step custom configuration example

Configurations are created and maintained with ESAcademy's CANopen Architect utility to edit Electronic Data Sheets (EDS). Example configurations are provided as part of the delivery.

Create a configuration

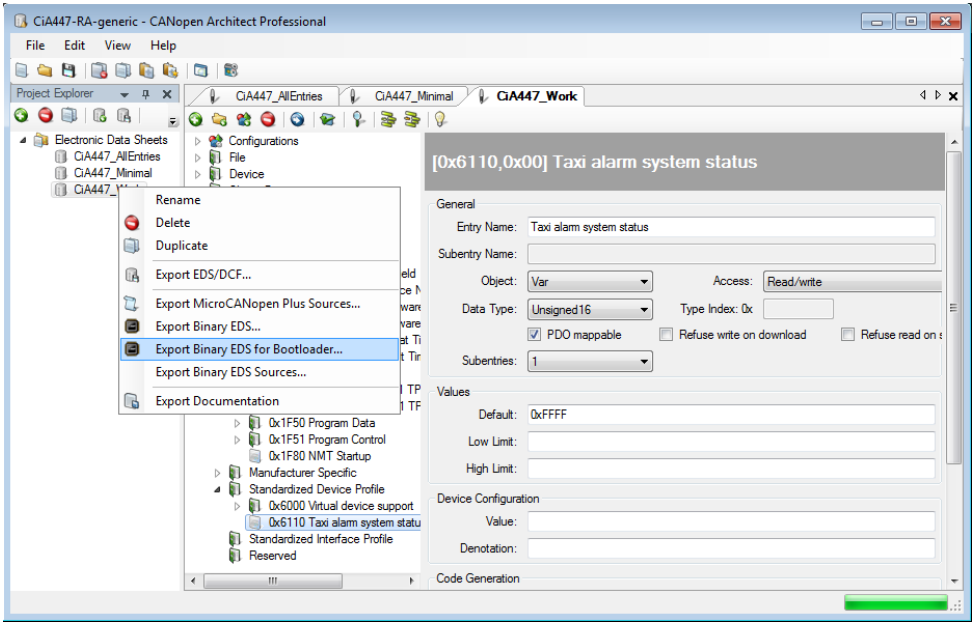
To start your own configuration, duplicate the "CiAxxx_Minimal" device and rename it to your own project. Review and edit the settings in the "File" and "Device" sections, as well as the object dictionary entries 1008h to 100Ah and 1018h.

Now add the entries that you still require for your device by copying them from the "CiAxxx_AllEntries" section or by adding them manually.

Export the configuration

To create the binary EDS configuration file required by CANopenIA devices, select "Export Binary EDS for Bootloader" from the local menu.

This creates a ".bin" file containing the binary EDS.



EXPORTING A CONFIGURATION FROM CANOPEN ARCHITECT

Load the configuration

The ".bin" file created can directly be transferred to the CANgineBerry using the COIAUpdater utility..

5 Minimal Manager SDO & PDO Handling

The CANopenIA Minimal Manager version simplifies how an application uses CANopen communication. Here all data is only referred to by a node ID and the Object (Index and Subindex) to address ab object in a node's object dictionary.

Using the write and read to a remote object dictionary functions, the host system can read and write all objects in a network.

In addition, the host system receives event notifications, if data came in from a remote object. Again, referred to by the node ID the data comes from and the object dictionary entry (Index and Subindex).

If your application is generic and does not require optimized communication (e.g. to lower the bus load for communication or achieve shorter reaction times), the this is all you need to know.

5.1 Communication options

The default CANopen communication mode used by the CANopenIA-MGR is the SDO communication (Service Data Objects). Here the manager sends one read/write request for a single object of a node and receives one response.

Internally, the manager scans detected devices for their PDO (Process Data Object) configuration. The scanned information is used by the minimal manger to configure itself for receiving all Transmit PDOs transmitted by the devices and for transmission of all Receive PDOs to the devices.

Receiving TPDO data from the devices

The application requires no knowledge about the Transmit PDO configuration of the devices. Once self-configured, the minimal manger receives all PDOs generated by the devices and converts them into the corresponding "New Data" indication events towards the host or application. The application automatically receives all PDO data.

For each object received, the host/application is informed about:

- The node ID which sent the data
- Which Object of that node was received (Index/Subindex)
- The data itself

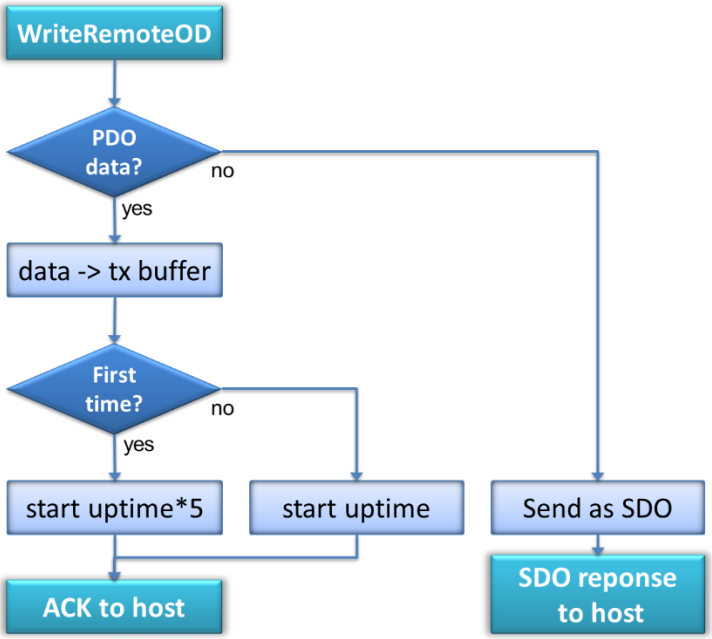
Sending data to the devices

The application addresses the data in the same fashion as for received data. It uses the WriteRemoteOD functionality and informs the CANopenIA Manager about:

- The node ID to which the data needs to be send
- Which Object of that node is it going to (Index/Subindex)
- The data itself

The manager automatically determines if this data can be send by PDO or if a SDO needs to be triggered. As PDOs can have multiple objects mapped (multiple object contained in one CAN message) all mapped items must be written at least once, before the PDO can be transmitted by the manager. This is required to prohibit transmission of uninitialized data/commands to a CANopen device.

We recommend that once the application receives the call-back that a mode has been scanned, it writes once to all objects of that device that can be written to, to ensure all data has been initialized.



PROCESSING REMOTE DATA WRITE REQUESTS

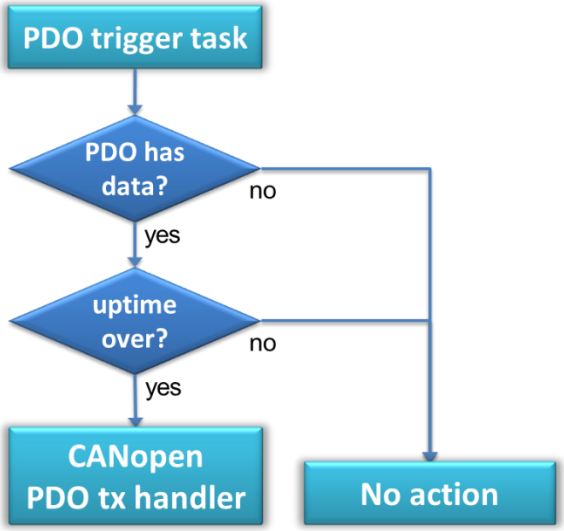
The flow chart above illustrates how the manager processes data write requests to nodes on the network. If the data written is not part of any PDO, it gets written to the

node using a SDO client write access. Once the SDO response comes back from the node, this response is passed on to the host.

Otherwise the data is copied to the appropriate buffer and the PDO update timer is handled. If it is the first use, then the timer value is multiplied with 5 to give the host more time to write all the data to init the PDO. The PDO gets queued/triggered ready for transmission once the PDO update timer expires.

Internally, the CANopenIA system continuously checks if a PDO requires transmission (see next flow chart “PDO trigger task”.

PDOs are only processed for transmission, if data is available and the update time expired.



PDO TRIGGER TASK

If both the default event and inhibit times (objects [5F01h,05h] and [5F01h,06h], see section 9.2) are zero, then a PDO is triggered for transmission whenever the update time expires (time since last write by the host) or the last mapped entry has been written.

Once set (non-zero), the event and inhibit times work as defined in CANopen:

If the inhibit time is set (non-zero) and the event time is zero, then any repetitive transmission is only transmitted, if the time since last transmission is at least as long as the inhibit time.

If the event time is set (non-zero) and the inhibit time is zero, then the PDO is transmitted cyclically, no matter if the data has been updated by the application or not.

If both times are set, then they are combined. If triggered by host (expiration of update time), then the PDO gets transmitted, observing the inhibit time. Without host triggering, the PDO is transmitted cyclically based on the event time.

6 The Remote Access Protocol

This chapter specifies the commands for controlling the CANopenIA Coprocessor via a serial interface. The protocol is suitable for tunneling through other networks such as a Bluetooth or TCP connection as well as for communication between a CANopen task and a host task within one system.

The communication between the host and the CANopenIA is based on messages with binary content and a check sum.

6.1 Definitions

Byte or UNSIGNED8:
8-bit, unsigned value

UNSIGNED16:
16-bit, unsigned value

UNSIGNED32:
32-bit, unsigned value

Host:
The processor or application controlling the CANopen CANopenIA via the interface specified in this document

Command:
Message from host to CANopenIA with a request to execute a command.

Response:
Message from CANopenIA to host in response to a command. Every command triggers a response. Some responses may take longer as CANopen communication might be involved. As a result one or multiple Indications might occur before receiving a response.

Indication:
Message from CANopenIA to host indicating the host that an event occurred.

Max data size:
In this version, the maximum user data size is 28 bytes. Including overhead, this results in a maximum serial packet size of 35 bytes.

Message Definition
Any message exchanged between Host and the CANopen node use the following structure (all Bytes):

<start character><length><command/response/indication><checksum>

Multi-Byte values are transmitted in little-endian format.

<start character> (Byte) default: 11h

1. Bits 0 to 3 indicate the network number, the value of zero is reserved, the default is one.
2. Bit 4 indicates if a checksum is used or not. If set, checksum is used, the default is one, using a checksum.
3. Bit 5 indicates if the length value has 8 or 16 bit. If set, 16 bits are used, the default is zero, using 8 bits for the length value.
4. Bits 6 to 7 are reserved.

<length> (Byte or UNSIGNED16, see Bit 5 of start character)

The total length of the command/response/indication in bytes.

<command/response/indication>

The data transferred in this packet can be a command, a response or an indication. For details see specifications below.

<checksum> (UNSIGNED16 or not used, see Bit 4 of start character)

A 16-bit CRC calculated with the Polynomial x16 + x15 + x2 + 1. The checksum calculation does not involve the start character.

6.2 Error Codes

Most of the responses contain an error code field. A value of zero means "no error". The bits in the error code field have the following meanings:

Bit	Meaning
0	Object Dictionary entry not found
1	Invalid command length
2	Invalid command
3	Busy (e.g. SDO client is currently in use)
4	No resources (e.g. internal problem obtaining an SDO
5	Transmit buffer is full
6	Transfer was aborted
7	Receive buffer size was too small
8	SDO toggle error
9	SDO timeout
10	Unknown/miscellaneous error
11	Not supported
12	Non-volatile memory write failure
13	Not all PDO mapped objects written

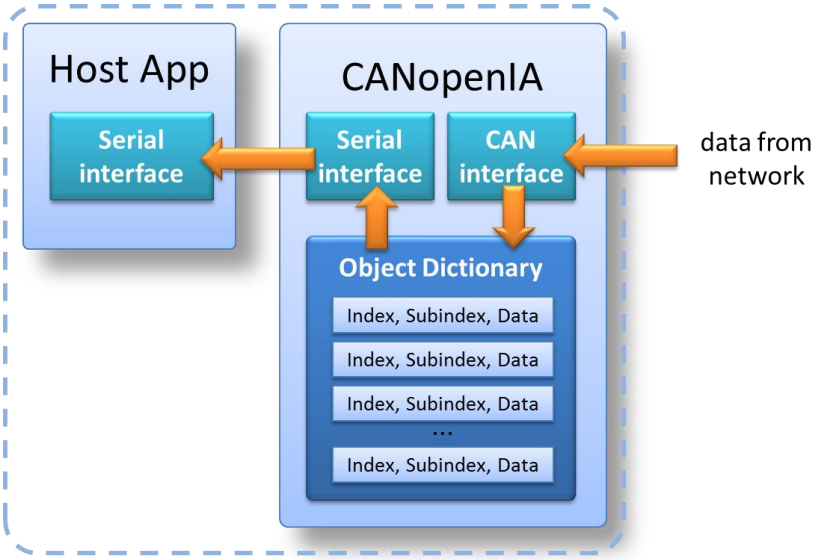
7 Commands, Responses and Indications

7.1 Access to local Object Dictionary

The commands, responses and indications of this section are used to access the local object dictionary of the CANopenIA Coprocessor.

The syntax shown is split into the serial version “Serial” (on lowest level, treated as an array of bytes) and the “C” style function interface, if used in the library version.

Indication "D": New process data written to local Object Dictionary



Automatic indication of new data written to Object Dictionary

New process data arrived from the CANopen network and was written to a local Object Dictionary entry. The node ID of the sender (if known), the Object Dictionary entry in question and the new data is part of this indication. This applies to both data received by SDO and PDO access. Data size is indicated via length field of lower communication layer or length parameter when use is a library.

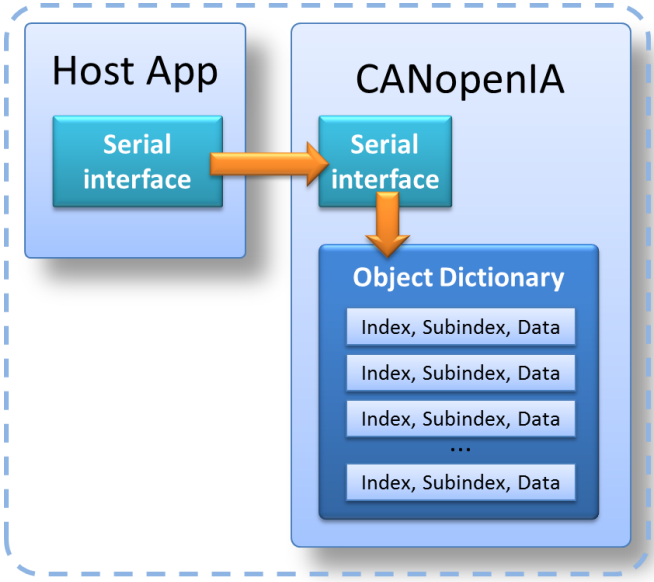
In minimal manager and CiA 447 mode ALL CiA defined/known PDOs are received and cause an indication. Advanced versions allow setting of optional filters to ignore unwanted PDOs.

Serial: `D<nodeid><index><subindex><data>`

Language	Prototype
C	`void NewData(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED16 length, UNSIGNED8 *data);`
C++	`static void NewData(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED16 length, UNSIGNED8 *data);`
Java	`public static void NewData(byte NodeID, int Index, byte Subindex, int DataLength, Pointer Data, Pointer Param);`

Parameter	Description
nodeid	The ID of the node sending the data
index	Index of the object dictionary entry in the node
subindex	Subindex of the object dictionary entry in the node
length	Length of data data
data	The data

Command "W": Write to a local Object Dictionary entry



Read or write command to local Object Dictionary

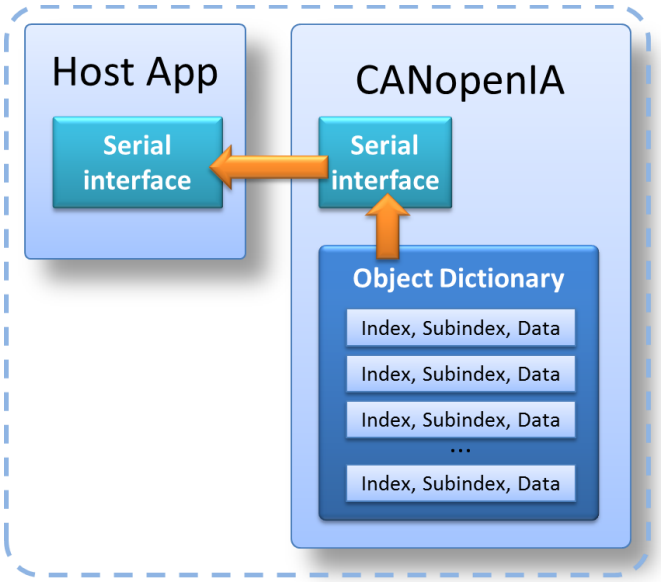
Writes data to one local Object Dictionary entry. Data size is indicated via length field of lower communication layer (see message definition).

Serial: `W<index><subindex><data>`

Language	Prototype
C	`UNSIGNED32 C_SerialProtocol_WriteLocalOD(UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`
C++	`UNSIGNED32 SerialProtocol::WriteLocalOD(UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`
Java	`long C_SerialProtocol_WriteLocalOD(short index, byte subindex, int datalength, Pointer data)`

Parameter	Description
index	The index of the object dictionary entry to write to
subindex	The subindex of the object dictionary entry to write to
datalength	Number of bytes to write
data	Data to write

Response "W": Write (local) response



Read or write response from local Object Dictionary

The following message is a response from the CANopen device to every “W” message processed.

Serial: `W<index><subindex><err>`

Not used in the programming interface *(WriteLocalOD is blocking and returns values)*

Command "R": Read from a local Object Dictionary entry

Request to read data from one Object Dictionary entry. Data size is indicated via length field of lower communication layer.

Serial: `R<index><subindex>`

Language	Prototype
C	`UNSIGNED32 C_SerialProtocol_ReadLocalOD(UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 *datalength, UNSIGNED8 *data)`
C++	`UNSIGNED32 SerialProtocol::ReadLocalOD(UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 *datalength, UNSIGNED8 *data)`
Java	`long C_SerialProtocol_ReadLocalOD(short index, byte subindex, Pointer datalength, Pointer data)`

Parameter	Description
index	The index of the object dictionary entry to read from
subindex	The subindex of the object dictionary entry to read from
datalength	When called set to the maximum number of bytes to read. On return holds the number of bytes read
data	Filled with read data

Response "R": Read (local) response

The following message is a response from the CANopen device to every “R” message processed. Data size is indicated via length field of lower communication layer (see message definition).

Serial: `R<index><subindex><err><data>`

Not used in the programming interface *(ReadLocalOD is blocking and returns values)*

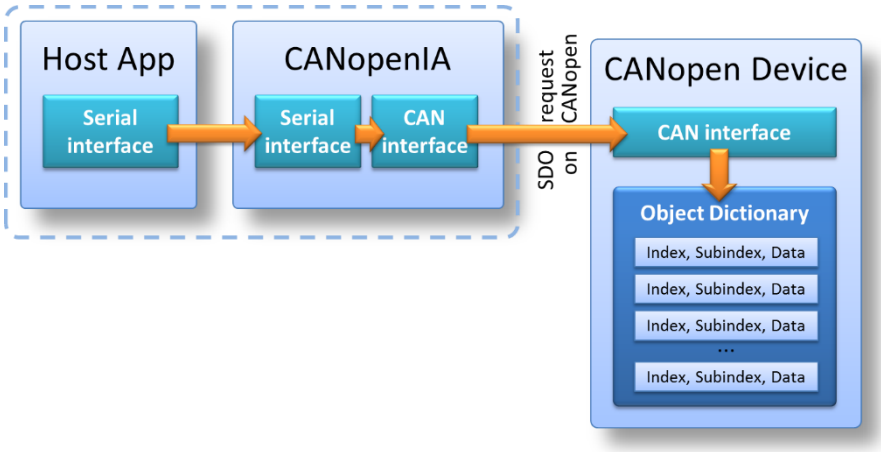
The return value is ERROR_NOERROR or an error code, ERROR_xxx.

7.2 Access to other nodes

The commands, responses and indications of this section are used to access object dictionary entries of any node on the network. In CANopen terminology these use SDO clients to communicate with the nodes addressed.

These commands require SDO clients which are only available when the Manager or CiA 447 functionality is enabled.

Command "S": Write to a remote Object Dictionary entry



Read or write request to a remote Object Dictionary of a node on the network

Writes data to one Object Dictionary entry of a remote node (using SDO client access). Data size is indicated via length field of lower communication layer.

Serial: `S<nodeid><index><subindex><data>`

Note: only one remote SDO operation can take place at a time. This applies to read and writes. An attempt to start a new SDO operation while one is still completing will generate an error.

Language	Prototype

C	`UNSIGNED32 C_SerialProtocol_WriteRemoteOD(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`
	
	`UNSIGNED32 C_SerialProtocol_WriteRemoteODExtended(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`

C++	`UNSIGNED32 SerialProtocol::WriteRemoteOD(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`
	
	`UNSIGNED32 SerialProtocol::WriteRemoteODExtended(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 datalength, UNSIGNED8 *data)`

Java	`long C_SerialProtocol_WriteRemoteOD(byte nodeid, short index, byte subindex, int datalength, Pointer data)`
	
	`long C_SerialProtocol_WriteRemoteODExtended(byte nodeid, short index, byte subindex, int datalength, Pointer data)`

Parameter	Description
nodeid	The ID of the node to write to
index	The index of the object dictionary entry to write to
subindex	The subindex of the object dictionary entry to write to
datalength	The number of bytes to write
data	Data to write

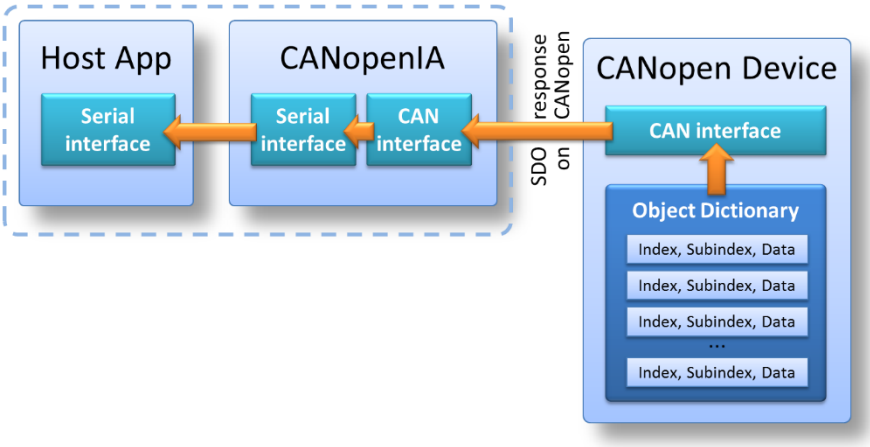
The “extended” versions are non-blocking. They return immediately and on completion of the write the SDO Request Complete callback function is called.

The return value is ERROR_NOERROR or an error code, ERROR_xxx.

Language	Prototype
C	`void SDORequestComplete(UNSIGNED8 nodeid, UNSIGNED32 result);`
C++	`static void SDORequestComplete(UNSIGNED8 nodeid, UNSIGNED32 result);`
Java	`public static void SDORequestComplete(byte NodeID, int Result);`

Parameter	Description
nodeid	The ID of the node that was written to
Result	The result of the write operation, SDOERR_OK or SDOERR_xxx

Response "S": Write (remote) response



Read or write response from a remote Object Dictionary of a node on the network

The following message is a response from the CANopen device to every “S” message processed.

Serial: `S<nodeid><index><subindex><err>`

Not used in the programming interface *(WriteRemoteOD is blocking and returns values)*

Command "U": Read from a remote Object Dictionary entry

Request to read data from a remote Object Dictionary entry (using SDO client access, upload).

Serial: `U<nodeid><index><subindex>`

Language	Prototype

C	`UNSIGNED32 C_SerialProtocol_ReadRemoteOD(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 * datalength, UNSIGNED8 *data)`
	
	`UNSIGNED32 C_SerialProtocol_ReadRemoteODExtended(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 * datalength, UNSIGNED8 *data)`

C++	`UNSIGNED32 SerialProtocol::ReadRemoteOD(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 *datalength, UNSIGNED8 *data)`
	
	`UNSIGNED32 SerialProtocol::ReadRemoteODExtended(UNSIGNED8 nodeid, UNSIGNED16 index, UNSIGNED8 subindex, UNSIGNED32 *datalength, UNSIGNED8 *data)`

Java	`long C_SerialProtocol_ReadRemoteOD(byte nodeid, short index, byte subindex, Pointer datalength, Pointer data)`
	
	`long C_SerialProtocol_ReadRemoteODExtended(byte nodeid, short index, byte subindex, Pointer datalength, Pointer data)`

Parameter	Description
nodeid	The ID of the node to read from
index	The index of the object dictionary entry to read from
subindex	The subindex of the object dictionary entry to read from
datalength	When called set to the maximum number of bytes to read. On return holds the number of bytes read
data	Filled with read data

The “extended” versions are non-blocking. They return immediately and on completion of the read the SDO Request Complete callback function is called.

The return value is ERROR_NOERROR or an error code, ERROR_xxx.

See “Write to a remote Object Dictionary entry command“ for details of the callback function.

8 Remote Access Application Example

As part of the delivery, a programming example is provided, it is named RemoteAcessApp and uses the serial commands and responses to access the CANopen network.

The remote access app is provided as command line executable for Linux and Windows along with all source files. It can be used as basis for own developments.

Use the -h parameter to get a list of supported command line parameters supported.

RA_App -h

Once connected, the RA_App displays all data received as follows:

{node ID, Index, Subindex, Length, Data}

Data received is displayed in hexadecimal along with the node ID from which the data was received (zero if unknwon), followed by the Index and Subindex indicators, followed by the data.

9 Object Dictionary entries in the manufacturer specific area

The manufacturer specific area of the Object Dictionary provides direct access to configuration data. These can be accessed using the read and write local commands. Syntax used in listing below:

Name

[index,subindex] (data type, access type)
Description

9.1 CANopenIA Device Status

The entries in this section give the host access to the current state of the local CANopenIA device. All these entries are read-only.

Device status: own node ID

[5F00h,01h] (UNSIGNED8, RO)
The node ID of the local CANopenIA device

Device status: own NMT state

[5F00h,02h] (UNSIGNED8, RO)
The current CANopen state of the local CANopenIA device. See section 9.3 for a list of all defined states.

Device status: own HW state

[5F00h,03h] (UNSIGNED8, RO)
Bit: 0: INIT – set after a completed initialization
1: CERR – set, if a CAN bit or frame error occurred
2: ERPA – set, if a CAN "error passive" occurred
3: RXOR – set, if a receive queue overrun occurred
4: TXOR – set, if a transmit queue overrun occurred
5: CANFD – set, if CAN hardware supports CAN FD
6: TXBSY – set, if Transmit queue is not empty
7: BOFF – set, if a CAN "bus off" error occurred

Device status: own HW/FW mode

[5F00h,04h] (UNSIGNED32, RO)
Bit: 0..7: 00h: Custom hardware
01h: CANgineBerry
02h: CANgineLight

03h: CANgineBT
04h: PCAN-RS232
05h: PCAN-xxx with PCAN-Basic API
Bit: 8..15: 00h: Custom firmware
01h: CANopenIA Device
02h: CANopenIA Manager
03h: CANopenIA 447izer
Bit: 16..23: Firmware major version
Bit: 24..31: Firmware minor version

Chip serial number (where available)

[5F00h,05h] (UNSIGNED128/DOMAIN, RO)

The serial number of the microcontroller hosting the CANopenIA software.

9.2 CANopenIA Device Control

The entries in this section can be written to and allow the host to actively control the local CANopenIA device or manager.

Device control: Reset

[5F01h,01h] (UNSIGNED8,WO)
Reset the CANopenIA chip, module or library. Writing129 issues a soft reset, 130 a hard reset.

Device control 447: Sleep Objection

[5F01h,02h] (UNSIGNED8,RW)
Activate the CiA 447 sleep objection (set to 1 to object).

Device control 447: Ignore PDOs from VD

[5F01h,03h] (UNSIGNED32,RW)
For If a bit is set in this value, then PDOs coming from the corresponding virtual device (see vdfg number in CiA-447) are ignored. For example: set bit 7 to ignore all PDOs coming from GPS devices.

Manager control (manager only)

[5F01h,04h] (UNSIGNED32,RW)
Bit: 0: KEEP_OP - set to keep nodes operational
(will send appropriate NMT command automatically)
1: HB receive all - set to activate automated heartbeat monitoring
(default HB times below are used)
2: PDO receive all - set to activate automated device TPDO handling

(scan devices for their transmit PDOs and receive them all)
3: PDO transmit all - set to activate automated device RPDO handling
(scan devices for their receive PDOs and produce them all)
4: Use scanned entries – set to activate caching of scanned entries.
If requested by host, reply from cache.
5: Enforce remote write SDO – set to enforce SDO write access when.
writing to an OD entry of a remote node, do not send as TPDO.
6: Manual TPDO trigger – set to activate manual transmit PDO triggering
(use [5F01h,0Ch] to trigger)
7-15: Reserved
16-22: Number of nodes supported for heartbeat monitoring and
SDO client handling
23: Reserved
24-30: Number of nodes supported for automated PDO handling
31: Reserved

Default heartbeat producer time (manager only)

[5F01h,05h] (UNSIGNED16,RW)
Use this default event time (in milliseconds) for all PDO transmissions by the manager.

Default heartbeat consumer time (manager only)

[5F01h,06h] (UNSIGNED16,RW)
Use this default event time (in milliseconds) for all PDO transmissions by the manager.

Default PDO update time (manager only)

[5F01h,07h] (UNSIGNED8,RW)
When the manager updates PDO transmission data, this update timeout is started before triggering the PDO for transmission. This allows the application to update all objects of a PDO before its transmission is triggered. Note that this time is not used, when the PDO event time (see below), is non-zero.

Default PDO transmission event time (manager only)

[5F01h,08h] (UNSIGNED16,RW)
Use this default event time (in milliseconds) for all PDO transmissions by the manager.

Default PDO transmission inhibit time (manager only)

[5F01h,09h] (UNSIGNED16,RW)

Use this default inhibit time (in 100th of microseconds) for all PDO transmissions by the manager.

Manager re-scan device (manager only)

[5F01h,0Ah] (UNSIGNED8,WO)
Writing a node ID to this entry re-triggers the auto-scan mechanism for this node. The manger will start a new node scan for this device.

Bit: 0-6: Node ID to scan
7: reserved

Device and Manager generic CAN Rx / Tx

[5F01h,0Bh] (UNSIGNED8,RW)

This feature enables the support of generic CAN messages, not handled by the local CANopen implementation.

Bit: 0-3: Generic CAN transmit
0: disabled
1: Condensed access via object 5F0Ch
4-7: Generic CAN receive
0: disabled
1: Condensed access via object 5F0Ch

CAN messages received, that are not processed by the local CANopen task are passed on to the host as a write to the local Object Dictionary entry 5F0Ch:

- Node ID: 0
- Index: 5F0Ch
- Subindex: Length of CAN message in bytes (0-8)
- Len: 2 + Length of CAN message in bytes (0-8)
- Data: First 2 byte: CAN ID
  Followed by the data bytes of the CAN message

To transmit a generic CAN message, execute a write to the local object dictionary entry 5F0Ch:

- Index: 5F0Ch
- Subindex: Length of CAN message in bytes (0-8)
- Len: 2 + Length of CAN message in bytes (0-8)
- Data: First 2 byte: CAN ID
  Followed by the data bytes of the CAN message

Testing with the COIA utility for the CANgineBerry

Activate the feature by a write to [5F01h,0Bh]:

```
-w 0x5F01,0x0B,0x01,0x11
```

To transmit a generic CAN message, use the “--tx-can” parameter, passing the CAN ID, the length and the data bytes:

```
--tx-can 0x150,4,0x11,0x22,0x33,0x44
```

This produces a 4-byte CAN message with ID 150h and the four data bytes 11h to 44h.

Use the monitoring “-m” parameter to monitor incoming data and messages.

Manual PDO trigger (device & manager)

[5F01h,0Ch] (UNSIGNED16,WO)

Manually trigger the transmission of a PDO. When using the CANopen device firmware (BEDS) version, the parameter passed is the Transmit PDO number to trigger, starting at 1.

For the manager (MGR) version, this is the only trigger method, if the automatic PDO triggering mechanism is disabled ([5F01h,04h] bit 6).

To trigger a PDO, write node ID and PDO number into this entry. The PDO triggered is the one matching the Receive PDO of that node.

Bit: 0-7: PDO number, starting at 1
8-14: Node ID, starting at 1
14: Reserved

Testing with the COIA utility for the CANgineBerry

Node number 5 is a digital I/O node (CiA 401)

```
-w 0x5F01,4,4,0x0000003F
--node-write 5,0x6000,1,1,0x55
--node-write 5,0x6000,2,1,0x66
--node-write 5,0x6000,3,1,0x77
--node-write 5,0x6000,4,1,0x88
-w 0x5F01,0x10,2,0x0501
```

In first line we enable automatic TPDO handling, the next four lines write data into the PDO and the last line triggers the PDO (node 5, PDO 1).

9.3 Status of all nodes

Only available with CANopenIA-MGR and 447 versions.

Last known state of Node 1

[5F04h,01h] (UNSIGNED8, RO)
The last known state of node 1, see list below for all defined values.

Last known state of Node X

[5F04h,X] (UNSIGNED8, RO)
The last known state of this node (allowed range 1 to 127), see list below for all defined values.

The following values are defined:

NODESTATUS_BOOT	0x00
NODESTATUS_STOPPED	0x04
NODESTATUS_OPERATIONAL	0x05
NODESTATUS_PREOP	0x7F
NODESTATUS_EMCY_NEW	0x80
NODESTATUS_EMCY_OVER	0x81
NODESTATUS_HBACTIVE	0x90
NODESTATUS_HBLOST	0x91
NODESTATUS_SCANSTARTED	0x9F
NODESTATUS_SCANCOMPLETE	0xA0
NODESTATUS_SCANABORTED	0xA8
NODESTATUS_RESETAPP	0xB0
NODESTATUS_RESETCOM	0xB1
NODESTATUS_SLEEP	0xF0
NODESTATUS_BOOTLOADER	0xF1

9.4 NMT Master Message

Only available with CANopenIA-MGR version. An NMT Master message can be triggered by writing to [5F0Ah,01h].

Transmit NMT (manager only)

[5F0Ah,01h] (UNSIGNED16, WO)
The high byte contains the destination node id (1-127) or zero for “all” nodes.
The low byte contains the NMT command:

01h: Switch to operational state
02h: Switch to stopped state
80h: Switch to pre-operational state
81h: Execute an application reset
82h: Execute a communication reset

9.5 Manager: Automatic Node Scan

In CANopen Manager or CiA 447 mode, the device automatically scans nodes found on the network for often used entries. This data is available, as soon as a node's state is reported as NODESTATUS_SCANCOMPLETE.

If caching is enabled in the Manager Control word (Object [5F01h,04h]), then the CANopenIA device will return the pre-scanned entries without re-requesting these from the device via CANopen.

Example: If the host requests the object [1018h,1] (vendor id) from node 3 by sending the ReadRemoteOD command, then the CANopenIA will directly reply with the value, if it is in the local cache.

10 The CAN232 Protocol Support

Some CANopenIA devices support selected commands from the CAN232 (ASCII) protocol by Lawicel. This is an ASCII protocol that can be directly used with many terminal programs. Each line ends with a CR (13) byte. Command lines send to the device are confirmed with a CR (13). In case of error, the device returns BELL (7).

If you do not want the CANopenIA device to start up with any CANopen style messages, then configure it to LSS mode and pre-select the CAN bitrate you want.

When CAN232 support is active, the red LED indicates continuous three flashes.

10.1 Open and Close CAN232 support

O[CR]

Use the single capital ‘O’ and ‘C’ to open or close the protocol support. While support is “open”, the green LED blinks, else it is solid green.

10.2 Changing the CAN bitrate

Sb

To change the CAN bitrate, send a ‘U’ followed by digit b from 4 to 8:

- 4 : 125 kbit/s
- 5 : 250 kbit/s
- 6 : 500 kbit/s
- 7 : 800 kbit/s
- 8 : 1 Mbit/s

10.3 Transmitting and receiving data

Use letter ‘t’ for 11-bit CAN ID, ‘T’ for 29-bit CAN ID.

Tiiildd..[CR], Tiiiiiiiildd..[CR]

i: hexadecimal number with 3 (for 11bit) or 8 digits with the CAN ID

l: 1 digit, 0 to 8 for the length of the message

d: data bytes, hexadecimal, 2 chars per byte, number of bytes matching previous parameter