# CAN-CBM-AI410

## 4 Analog Inputs

CANopen-
Software Manual

| | |
|---|---|
| **Manual file:** | I:\texte\Doku\MANUALS\CAN\CBM\AI410\Englisch\AI410_13S.en9 |
| **Date of print:** | 22.02.2005 |

| | |
|---|---|
| **Software described:** | CANopen |
| **Revision:** | 1.1 |

**Changes in the chapters**

The changes in the manual listed below affect changes in the **firmware** as well as changes in the **description** of facts only.

| Chapter | Changes versus previous version |
|---|---|
| 1.5 | Chapter “Conversion of the Analog Values” inserted |
| - | - |

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

**esd** assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd** gmbh.

**esd** does not convey to the purchaser of the product described herein any license under the patent rights of **esd** gmbh nor the rights of others.

**esd electronic system design gmbh**
Vahrenwalder Str. 207
30165 Hannover
Germany

Phone: +49-511-372 98-0
Fax: +49-511-372 98-68
E-mail: info@esd-electronics.com
Internet: www.esd-electronics.com

**USA / Canada:**
**esd electronics Inc.**
12 Elm Street
Hatfield, MA 01038-0048
USA

Phone: +1-800-732-8006
Fax: +1-800-732-8093
E-mail: us-sales@esd-electronics.com
Internet: www.esd-electronics.us

## Contents

**Page**

This page has intentionally been left blank.

# 1. General

This chapter contains fundamentals on CANopen. A complete CANopen description would exceed the limits of this manual. Further information can therefore be taken from the CAL / CANopen - documentation ‘CiA Draft Standard 201...207’, ‘CiA Draft Standard 301’ and ‘CiA Draft Standard Proposal 401’.

## 1.1 Definition of Terms

| | |
|---|---|
| COB ... | Communication Object |
| Emergency-ID... | Emergency Data Object |
| n/a ... | not available |
| NMT... | Network Management (Master) |
| Rx... | receive |
| SDO... | Service Data Object |
| SW ... | Software |
| Sync... | Sync(frame) Telegram |
| tbd ... | to be defined (data has yet to be defined) |
| Tx... | transmit |
| Node-ID | CANopen-module number |

## 1.2 NMT-BOOT-Up

### 1.2.1 Minimum Capability Device Boot-up

The CAN-CBM-AI410-module can be initialized with the 'Minimum Capability Device' boot-up as described in CiA-Draft Standard 301.

Usually a telegram to switch from *Pre-Operational* status to *Operational* status after boot-up is sufficient. For this the 2-byte telegram '$01_h$', '$00_h$', for example, has to be transmitted to CAN-identifier '$0000_h$' (= Start Remote Node all Devices).

## 1.3 CANopen-Object Directory

The object directory is mainly a (sorted) group of objects which can be accessed via the network. Each object in this directory is addressed with a 16-bit index. The index in the object directories is represented in hexadecimal format.

The index can be a 16-bit parameter in accordance with the CANopen specification (CiA-Draft DS301) or a manufacturer-specific code. By means of the MSB of the index the object class of the parameter is defined.

Part of the object directory are among others:

| Index [Hex] | Object | Example |
|---|---|---|
| 0001 ... 009F | definition of data types | |
| 1000 ... 1FFF | Communication Profile Area | $1001_h$ = error register |
| 2000 ... 5FFF | Manufacturer Specific Profile Area | |
| 6000 ... 9FFF | Standardised Device Profile Area | in accordance with application profile DS401 |
| A000 ... FFFF | reserved | |

PDOs (Process Data Objects)

PDOs are used to transmit process data.
In the 'Receive'-PDO process data from the CAN-CBM-AI410-module are received.
In the 'Transmit'-PDO the CAN-CBM-AI410-module (which is always a SLAVE in the CANopen network!) transmits data on the CANopen network.

The asynchronous transmission mode of the PDOs is defined by 'PDO transmission type' in the communication parameter of the according process data object.
A synchronous transmission is not yet being supported.

## 1.4 Communication Parameters of the PDOs

The communication parameters of the PDOs (in accordance with DS301) are transmitted as SDO (Service Data Objects) on ID '**$600_h$ + Node-ID**' (request). The receiver acknowledges the parameters on ID '**$580_h$ + Node-ID**' (response).
The **Node-ID** (module number) is set via the coding switches (SW100, SW101). For possible settings please refer to the hardware manual of the CAN-CBM-AI410-module.

### 1.4.1 Accessing the Communication Parameters via SDO-Telegrams

SDOs (Service-Data Objects) are used to access the object directory of a device.
An SDO therefore is a 'channel' to access the parameters of the device. The access via this channel is possible in *Operational* and *Preoperational* status with the CAN-CBM-AI410-module.

This chapter does not show every possible, but only a few important access modes for the CAN-CBM-AI410-module.
Definitions for the access modes can be taken from CiA DS202-2 (CMS-Protocol Specification, chapter 6: CMS Multiplexed Domain Protocols).

An SDO is structured as represented in the table below:

| Identifier | Command code | Index (low) | Index (high) | Sub-index | LSB | Data field | MSB |
|---|---|---|---|---|---|---|---|

**Example :**
Requesting the device type for the module by means of Node-ID 19 ($13_h$)
Request:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $600_h$ + Node-ID = $613_h$ | $40_h$ (read) | $00_h$ | $10_h$ (Index = $1000_h$) (Receive-PDO-Comm-Para) | $00_h$ | $00_h$ | $00_h$ | $00_h$ | $00_h$ |

Response:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $580_h$ + Node-ID = $593_h$ | $43_h$ (read) | $00_h$ | $10_h$ (Index = $1000_h$) (Receive-PDO-Comm-Para) | $00_h$ | $91_h$ | $01_h$ | $02_h$ | $00_h$ (Device type = $0002.0191_h$) |

The value of the device type for the module with Node-ID 19 ($13_h$) is: $0002.0191_h$

**Identifiers**
The parameters are transmitted on ID '**$600_h$ + Node-ID**' (request).
The receiver acknowledges the parameters on ID '**$580_h$ + Node-ID**' (response).

### Command Code

The command code transmitted is combined from the command specifier and the length. Frequently required combination are, for instance:

$40_h = 64_d$: Read Request, i.e. a parameter is to be read
$23_h = 35_d$: Write Request with 32-bit data, i.e. a parameter is to be set.

The CAN-CBM-AI410-module responds to every received telegram with a response telegram. This can contain the following command codes:

$43_h = 67_d$: Read Response, this telegram contains the desired parameter
$60_h = 96_d$: Write Response, i.e. a parameter has been set successfully
$80_h = 128_d$: Error Response, i.e. the CAN-CBM-AI410-module reports a communication error.

### Frequently used Command Codes

The following table gives an overview of frequently used command codes. The command frames must always contain 8 data bytes. Notes on the syntax and further command codes can be found in CiA DS202-2 (CMS-Protocol Specification, chapter 6: CMS Multiplexed Domain Protocols).

| Command | for number of data bytes | Command code [Hex] |
|---|---|---|
| Write Request (Initiate Domain Download) | 1<br>2<br>3<br>4 | 2F<br>2B<br>27<br>23 |
| Write Response (Initiate Domain Download) | - | 60 |
| Read Request (Initiate Domain Upload) | - | 40 |
| Read Response (Initiate Domain Upload) | 1<br>2<br>3<br>4 | 4F<br>4B<br>47<br>43 |
| Error Response (Abort Domain Transfer) | - | 80 |

**Error Codes of SDO-Domain Transfer**

The following error codes might occur (in accordance with CiA Work Draft WD-301):

| Error code [Hex] | CAN-CBM-AI410-designation | Description |
|---|---|---|
| 0x05030000 | SDO_TOGGLE_BIT_NOT_ALTERNATED | toggle bit not changed |
| 0x05040001 | SDO_CS_UNKNOWN | wrong command specifier |
| 0x06010002 | SDO_READ_ONLY | write access not permissible |
| 0x06020000 | SDO_WRONG_INDEX | wrong index |
| 0x06060000 | SDO_HARWARE_ERROR | no access due to hardware error |
| 0x06070010 | SDO_WRONG_LENGTH | wrong number of data bytes |
| 0x06070012 | SDO_PARA_TO_LONG | service parameter too long |
| 0x06070013 | SDO_PARA_TO_SHORT | service parameter too short |
| 0x06090011 | SDO_WRONG_SUBIND | wrong sub-index |
| 0x06090030 | SDO_VALUE_RANGE_EXEEDED | - |
| 0x08000000 | SDO_OTHER_ERROR | undefined error source |
| 0x08000020 | SDO_DATA_CANNOT_STORED | data cannot be transferred or stored |

**Index, Sub-Index**
Index and sub-index will be explained in the following chapter ‘Communication Profile Area’.

**Data Field**
The data field has got a size of a maximum of 4 bytes and is always structured ‘LSB first, MSB last’. The least significant byte is **always** in ‘Data 1’.
With 16-bit values the most significant byte (bits 8...15) is always in ‘Data 2’, and with 32-bit values the MSB (bits 24...31) is always in ‘Data 4’.

## 1.5 Conversion of the Analog Values

The A/D-values are cyclically converted every 2 ms.

The last converted A/D-values are transmitted:

- if the module has received the number of SYNC-Messages (1...240), defined in *transmission type* in object $1800_h$
- if a remote request for a TxPDO is received
- if an interrupt is triggered by the change of an A/D-value (set with object $6426_h$)

This page has intentionally been left blank.

# 2. Communication Profile Area

## 2.1 Common Terms and Abbreviations

The following terms will be used in the tables representing the communication parameters:

| | |
|---|---|
| PDO-Mapping | PDO-mapping is available for this sub-index of the PDO |
| Access Mode | permissible access modes to this parameter<br>ro... read_only<br>This parameter can only be read. Write accesses cause an error message.<br>const.... constant<br>This parameter cannot be changed by the user. It can only be read. Write accesses cause an error message.<br>rw... read&write<br>This parameter can be read or set. |
| Value Range | value range of the parameter |
| Default value | default setting of parameter when module leaves the manufacturer |
| Name/Description | name and short description of parameter |

## 2.2 Overview of Communication Parameters

The format of the communication parameters can be taken from CiA DS301. Special features of implementing the CAN-CBM-AI410 will be described in the following chapters of this manual.

The module only supports the communication parameters listed in the table below.

| Index [Hex] | Name | Sub- | Type | Access | Default |
|---|---|---|---|---|---|
| 1000 | Device Type | - | Unsigned32 | ro | $00040191_h$ |
| 1001 | Error Register | - | Unsigned8 | ro | Error-Code |
| 1002 | Status Register | - | Unsigned32 | ro | 0 |
| 1003 | Pre-Defined Error Field | 16 | Unsigned32 | rw | Index 0 |
| 1005 | COB ID Sync | - | Unsigned32 | rw | $80_h$ |
| 1008 | Manufacturer's Device Name 1*) | - | Visible String 1*) | ro | CAN-CBM-AI410 3*) |
| 1009 | Manufacturer's Hardware Version | - | Visible String | ro | 11 |
| 100A | Manufacturer's Software Version | - | Visible String 2*) | ro | 10 |
| 100C | Guard Time | - | Unsigned16 | rw | 0 sec |
| 100D | Life Time Factor | - | Unsigned8 | rw | 0 |
| 1010 | Store Parameter | 0, 1, 2, 3 | Unsigned32 | rw | - |
| 1011 | Restore Parameter | 0, 1, 2, 3 | Unsigned32 | rw | - |
| 1014 | COB-ID Emergency message | - | Unsigned32 | rw | $80_h$ + Node-ID |
| 1017 | Producer Heartbeat Time | - | Unsigned16 | rw | 0 ms |
| 1018 | Identity Object | 4 | Unsigned32 | ro | $00000017_h$ |
| 1020 | Verify Configuration | 0, 1, 2 | Unsigned32 | rw | 0 |
| 1029 | Error Behaviour | 0, 1 | Unsigned32 | rw | 0 |
| 1800 | Transmit PDO Communication | 0,1,2,5 | PDOCommPar | -- | -- |
| 1A00 | Transmit PDO Mapping Parameter | 0,...5 | PDOMapping | ro | -- |

ro - Read Only, rw - Read/Write
1*) Attention: length > 4 bytes, i.e. upload more complicated than with other parameters (see page 17)
2*) depending on software revision, 3*) depending on default configuration

## 2.3 Description of Parameters

### 2.3.1 Device Type $1000_h$

| **INDEX** | **$1000_h$** |
|---|---|
| Name | *device type* |
| Data Type | unsigned 32 |
| Default Value | No |

The value of *device type* is: $0004.0191_h$ (analog output: $0004_h$
device profile number: $0191_h$ )

**Example: Reading-out the Device Type**

The CANopen master transmits the read request to the CAN-CBM-AI410-module with module number 3 (Node-ID = $3_h$) by means of identifier '$603_h$':

| ID | RTR | LEN | DATA | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| $603_h$ | $0_h$ | $8_h$ | $40_h$ | $00_h$ | $10_h$ | $00_h$ | $00_h$ | $00_h$ | $00_h$ | $00_h$ |
| | | | Read Request | Index=$1000_h$ | | Sub Index | | | | |

The CAN-CBM-AI410-module with Node-ID 3 responds to the master by means of the read response via identifier '$583_h$' (580 + Node-ID) the value of the device type:

| ID | RTR | LEN | DATA | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| $583_h$ | $0_h$ | $8_h$ | $43_h$ | $00_h$ | $10_h$ | $00_h$ | $91_h$ | $01_h$ | $04_h$ | $00_h$ |
| | | | Read Response | Index=$1000_h$ | | Sub Index | Dig. Profile No.191 | | Analog Input | |

value of the device type: $0004.0191_h$

The data field is always structured 'LSB first, MSB last' (see chapter 1.4.1, Data Field).

### 2.3.2 Error Register $1001_h$

The CAN-CBM-AI410-module uses the error register in order to show error messages.

| **INDEX** | **$1001_h$** |
|---|---|
| Name | *error register* |
| Data Type | unsigned 8 |
| Default Value | No |

The following bits of the error register are currently being supported:

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Meaning | - | - | - | - | - | - | - | *generic* |

The bits not being supported (-) are always returned as ‘0’.

| *generic* | Error |
|---|---|
| 0 | no error |
| 1 | generic error |

The following messages are available:
$00_h$ no error
$01_h$ generic error

### 2.3.3 Manufacturer Status Register $1002_h$

| **INDEX** | **$1002_h$** |
|---|---|
| Name | *manufacturer status register* |
| Data Type | unsigned 32 |
| Default Value | No |

The bits of the status register are assigned as represented below:

| Register bit (Assembler) | Description | Level assignment | |
|---|---|---|---|
| D25...D31 | not assigned | 0 | (always ‘0’) |
| D24 | I²C-error | 0<br>1 | no I²C-error<br>I²C-error |
| D3...D23 | not assigned | 0 | (always ‘0’) |
| D2 | watchdog reset | 0<br>1 | no reset<br>watchdog reset has been triggered |
| D1 | nodeguard enabled | 0<br>1 | nodeguarding disabled<br>nodeguarding enabled |
| D0 | heartbeat enabled | 0<br>1 | heartbeat function disabled<br>heartbeat function enabled |

### 2.3.4 Pre-defined Error Field $1003_h$

| **INDEX** | **$1003_h$** |
|---|---|
| Name | *pre-defined error field* |
| Data Type | unsigned 32 |
| Default Value | No |

In the *pre-defined error field* a list of the errors which occurred last is stored. Sub-index 0 contains the current number of errors stored in the list. Under sub-index 1 the last error which occurred is stored. If a new error occurs, the previous error is stored under sub-index 2 and the new error under sub-index 1, etc. In this way a list of the error history is created.

The error buffer is structured like a ring buffer. If it is full, the oldest entry is deleted for the latest entry.

This module supports a maximum of 16 error entries. When the 17th error occurs the oldest error entry is deleted. In order to delete the entire error list, sub-index '0' has to be set to '0'. This is the only permissible write access to the object.

With every new entry to the list the module transmits an **Emergency Frame** to report the error.

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Default [Dec] | Data type | R/W |
|---|---|---|---|---|---|---|
| **1003** | 0 | *no_of_errors_in_list* | 0, 1...10 | - | unsigned 8 | rw |
| | 1 | *error-code n* | 0...FFFFFFFF | - | unsigned 32 | ro |
| | 2 | *error-code (n-1)* | 0...FFFFFFFF | - | unsigned 32 | ro |
| | : | *:* | : | : | : | ro |
| | 16 | *error-code (n-15)* | 0...FFFFFFFF | - | unsigned 32 | ro |

Meaning of variables:

*no_of_errors_in_list* - contains the number of error codes currently on the list
*n* = number of error which occurred last
- in order to delete the error list this variable has to be set '0'
- if *no_of_errors_in_list* ≠0, the error register (Object $1001_h$) is set

*error-code x* The 32-bit long error code is combined from the CANopen-Emergency-Error-Code described in DS-301, Table 21 and the error code defined by esd (Manufacturer-Specific Error Field).

| Bit: | 31 ... ... 16 | 15 ... ... 0 |
|---|---|---|
| Contents: | *manufacturer-specific error field* | *emergency-error-code* |

*manufacturer-specific error field*: with CAN-CBM-AI410 always '00'

*emergency-error-code*: only the following error codes are being supported:
$0000_h$ - Emergency-Error-Reset or no Error
$1000_h$ - Emergency Generic Error
(error of I²C-EEPROM)

**Emergency Frame**

The data of the emergency frame transmitted by the CAN-CBM-AI410 is structured as follows:

| Byte: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Contents: | *emergency-error-code* (see above) | | *error-register* $1001_h$ | *no_of_errors_in_list* $1003,01_h$ | *manufacturer status register* $1002_h$ | | | |

### 2.3.5 COB-ID of SYNC-Message $1005_h$

| **INDEX** | **$1005_h$** |
|---|---|
| Name | *COB-ID SYNC message* |
| Data Type | unsigned 32 |
| Default Value | $80_h$ |

Structure of parameter:

| Bit No. | Value | Meaning |
|---|---|---|
| 31 (MSB) | - | do not care |
| 30 | 0/1 | 0: no SYNC generation<br>1: module generates SYNC |
| 29 | 0 | always 0, because 11-bit ID |
| 28...11 | 0 | always 0, because 29-bit Ids are not being supported |
| 10...0 (LSB) | x | bit 0...10 of SYNC-COB-ID |

The identifier can take values between $0...7FF_h$.

### 2.3.6 Manufacturer's Device Name $1008_h$

| **INDEX** | **$1008_h$** |
|---|---|
| Name | *manufacturer's device name* |
| Data Type | visible string |
| Default Value | string: 'CAN-CBM-AI410' |

When uploading the string the data has to be transferred as represented in the table below (all values in hexadecimal format). Eight data bytes must always be transmitted in the transferred commands.

| | Byte 0 | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Note |
|---|---|---|---|---|---|---|---|---|---|
| Client: | **40** | **08** | **10** | 00 | 00 | 00 | 00 | 00 | upload request index $1008_h$, sub-index=0 |
| CBM-AI410: | **41** | **08** | **10** | **00** | **0C** | 00 | 00 | 00 | upload ackn., len = 12 |
| Client: | **60** | 00 | 00 | 00 | 00 | 00 | 00 | 00 | segment upload request |
| CBM-AI410: | **00** | **43** | **41** | **4E** | **2D** | **43** | **42** | **4D** | 'CAN-CBM', len = 7 |
| Client: | **70** | 00 | 00 | 00 | 00 | 00 | 00 | 00 | - |
| CBM-AI410: | **15** | **2D** | **41** | **49** | **34** | **31** | **30** | 00 | '-AI410', len = 5, ended |

For a detailed description of the domain upload please refer to CiA DS202-2 (CMS-Protocol Specification).

### 2.3.7 Manufacturer's Hardware Version $1009_h$

| **INDEX** | **$1009_h$** |
|---|---|
| Name | *manufacturer's hardware version* |
| Data Type | visible string |
| Default Value | string: e.g '1.1' |

The hardware version is read similarly to reading the manufacturer device name via the domain upload protocol. Please refer to CiA DS202-2 (CMS-Protocol Specification) for a detailed description of the upload.

### 2.3.8 Manufacturer's Software Version $100A_h$

| **INDEX** | **$100A_h$** |
|---|---|
| Name | *manufacturer's software version* |
| Data Type | visible string |
| Default Value | string: e.g: '1.0' |

Reading the software version is similar to reading the manufacturer device name via the domain upload protocol. Please refer to CiA DS202-2 (CMS-Protocol Specification) for a detailed description of the upload.

### 2.3.9 Guard Time $100C_h$ and Life Time Factor $100D_h$

The CAN-CBM-AI410-module supports the node guarding or alternatively the heartbeat function (see page 24).
Guard time and life time factors are evaluated together. Multiplying both values will give you the life time. The guard time is represented in milliseconds.

| **INDEX** | **$100C_h$** |
|---|---|
| Name | *guard time* |
| Data Type | unsigned 16 |
| Default Value | 0 [ms] |
| Minimum Value | 0 |
| Maximum Value | $FFFF_h$ (65,535 s) |

| **INDEX** | **$100D_h$** |
|---|---|
| Name | *life time factor* |
| Data Type | unsigned 8 |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | $FF_h$ |

### 2.3.10 Store Parameters $1010_h$

Via this command the parameters are stored in the EEPROM. Only command 'Save all Parameters' is being supported.

When writing the index, the byte sequence represented below has to be transmitted. Reading the index returns information about the implemented storage functions (for more details refer to CiA DS-301).

| **INDEX** | **$1010_h$** |
|---|---|
| Name | *store parameters* |
| Data Type | unsigned 32 |

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Data type | R/W |
|---|---|---|---|---|---|
| **1010** | 0 | *number_of_entries* | $3_h$ | unsigned 8 | ro |
| | 1 | *save_all_parameters* | no default, write: 65 76 61 73 h (= ASCII: 'e' 'v' 'a' 's') | unsigned 32 | rw |
| | 2 | *save_communication_parameter* | no default, write: 65 76 61 73 h (= ASCII: 'e' 'v' 'a' 's') | unsigned 32 | rw |
| | 3 | *save_application_parameter* | no default, write: 65 76 61 73 h (= ASCII: 'e' 'v' 'a' 's') | unsigned 32 | rw |

Parameters which can be stored or loaded:

Communication parameters:

- $1005_h$ *COB-ID Sync*
- $100C_h$ *Guard Time*
- $100D_h$ *Life Time Factor*
- $1017_h$ *Producer Heartbeat Time*
- $1020_h$ Verify Configuration: *Configuration_Date, Configuration_Time*
- $1029_h$ Error Behaviour Object: *Communication_Error*

Application parameters:

- $6421_h$ Interrupt Trigger Selection: *Analog_Input_1, Analog_Input_2*, *Analog_Input_3, Analog_Input_4*
- $6426_h$ Analog Input Interrupt Delta unsigned: *Analog_Input_1, Analog_Input_2*, *Analog_Input_3, Analog_Input_4*

### 2.3.11 Restore Default Parameters $1011_h$

Via this command the default parameters, valid when leaving the manufacturer, are activated again. Every individual setting stored in the EEPROM will be lost. Only command 'Restore all Parameters' is being supported.

When writing the index the byte sequence represented below has to be transmitted. Reading the index returns information about the implemented restore function (for details please refer to CiA DS-301).

| **INDEX** | **$1011_h$** |
|---|---|
| Name | *restore default parameters* |
| Data Type | unsigned 32 |

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Data type | R/W |
|---|---|---|---|---|---|
| **1011** | 0 | *number_of_entries* | 3 | unsigned 8 | ro |
| | 1 | *load_all_default_parameters* | no default, write: 64 61 6F 6C h (= ASCII: 'd' 'a' 'o' 'l') | unsigned 32 | rw |
| | 2 | *load_communication_parameter* | no default, write: 64 61 6F 6C h (= ASCII: 'd' 'a' 'o' 'l') | unsigned 32 | rw |
| | 3 | *load_application_parameter* | no default, write: 64 61 6F 6C h (= ASCII: 'd' 'a' 'o' 'l') | unsigned 32 | rw |

The communication parameters and the application parameters which can be stored or loaded are listed in object $1010_h$ (see page 21).

### 2.3.12 COB_ID Emergency Object $1014_h$

| **INDEX** | **$1014_h$** |
|---|---|
| Name | *COB-ID emergency object* |
| Data Type | unsigned 32 |
| Default Value | $80_h$ + Node-ID |

This object determined the COB-ID of the emergency object (EMCY).

The structure of the object is represented in the table below:

| Bit No. | Value | Meaning |
|---|---|---|
| 31 (MSB) | 0/1 | 0: EMCY exists / is valid<br>1: no EMCY / EMCY is not valid |
| 30 | 0 | reserved (always 0) |
| 29 | 0 | always 0, because 11-bit ID |
| 28...11 | 0 | always 0, because 29-bit IDs are not being supported |
| 10...0 (LSB) | x | bit 0...10 of COB-ID |

The identifier can adapt values between $0...7FF_h$.

### 2.3.13 Producer Heartbeat Time $1017_h$

| **INDEX** | **$1017_h$** |
|---|---|
| Name | *producer heartbeat time* |
| Data Type | unsigned 16 |
| Default Value | 0 ms |

Here you specify the time with which the CAN-CBM-AI410-module cyclically transmits a heartbeat frame on the node-guarding ID.

If a value larger than zero is set for producer-heartbeat time, it is active and suppresses the node-/life guarding (see page 32).
If the producer-heartbeat time is set '0', the transmission of the heartbeat by this module is terminated.

To make the CANopen modules monitor each other (especially to detect connection failures), the heartbeat function can be used. In contrast to node guarding/life guarding the heartbeat function does not require RTR-frames.

**Function:**
A module, the so-called heartbeat producer, cyclically transmits a heartbeat message on the CAN-bus on the node-guarding identifier (see object $100E_h$). One or more heartbeat consumer receive the message. It has to be received within the heartbeat time stored on the heartbeat consumer, otherwise a heartbeat event is triggered on the heartbeat-consumer module.

Each module can act as a heartbeat producer and a heartbeat consumer. One CAN-network can contain several heartbeat producers and heartbeat consumers. The CAN-CBM-AI410-module only operates as a heartbeat producer.

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Default [Dec] | Data type | R/W |
|---|---|---|---|---|---|---|
| **1017** | 0 | *producer-heartbeat_time* | 0...FFFF | 0 ms | unsigned 16 | rw |

*producer-heartbeat_time* Cycle period of heartbeat producer to transmit the heartbeat on the node-guarding ID (see object $100E_h$).
The consumer-heartbeat time of the monitoring module must always be larger than the producer-heartbeat time of the heartbeat-transmitting module.

### 2.3.14 Identity Object $1018_h$

| **INDEX** | **$1018_h$** |
|---|---|
| Name | *identity object* |
| Data Type | unsigned 32 |
| Default Value | No |

The identity object contains general information about the CAN-module.

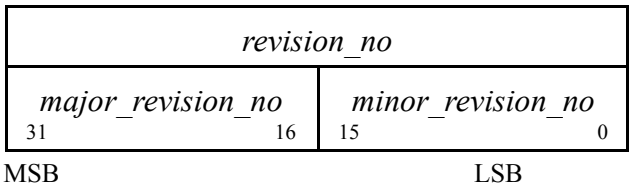
| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Default [Hex] | Data type | R/W |
|---|---|---|---|---|---|---|
| **1018** | 0 | *no_of_entries* | 1 | 1 | unsigned 8 | ro |
| | 1 | *vendor_id* | 0...FFFFFFFF | 0000 0017 | unsigned 32 | ro |
| | 2 | *product_code* | 0...FFFFFFFF | 2283 5020 | unsigned32 | ro |
| | 3 | *revision_number* | 0...FFFFFFFF | 'current SW-version' | unsigned32 | ro |
| | 4 | *serial_number* | 0...FFFFFFFF | 'current HW-version' | unsigned32 | ro |

Meaning of variables:

*vendor_id* This variable contains the esd-vendor-ID. This is always $00000017_h$.

*product_code* Here the esd-article number of the product is stored.
Example:
Value '2**2835.02**0h' corresponds to article number 'C.2835.02'.

*revision_number* Here the software version is stored. In accordance with DS-301 the two MSB represent the revision numbers of the major changes and the two LSB show the revision number of minor corrections or changes.



*serial_number*

Here the serial number of the hardware is read. The first two characters of the serial number are letters which designate the manufacturing lot. The following characters represent the actual serial number.

In the two MSB of *serial_no* the letters of the manufacturing lot are coded. They each contain the ASCII-code of the letter with the MSB set '1' in order to be able to differentiate between letters and numbers:
(ASCII-Code) + 80h = read_byte

The two LSB contain the number of the module as BCD-value.

Example:
If the value 'C1C2 0105h' is being read, this corresponds to the hardware-serial number code 'AB 0105'. This value has to correspond to the serial number of the module.

### 2.3.15 Verify Configuration $1020_h$

| **INDEX** | **$1020_h$** |
|---|---|
| Name | *verify configuration* |
| Data Type | unsigned 32 |
| Default Value | No |

In this object date and time of the last configuration can be stored in order to be able to check at a later time, whether the configuration stored corresponds to the one expected.

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Default [Dec] | Data type | R/W |
|---|---|---|---|---|---|---|
| **1020** | 0 | *no_of_entries* | 2 | 2 | unsigned 8 | ro |
| | 1 | *configuration_date* | 0...FFFFFFFF | 0 | unsigned 32 | rw |
| | 2 | *configuration_time* | 0...FFFFFFFF | 0 | unsigned 32 | rw |

Meaning of variables:

*configuration_date* Date of last configuration of module, represented in days since the 01.01.1984.

*configuration_time* Time in ms since midnight of the day of the last configuration.

### 2.3.16 Error Behaviour Object $1029_h$

| **INDEX** | **$1029_h$** |
|---|---|
| Name | *error behaviour object* |
| Data Type | Byte |
| Default Value | No |

If an error event occurs (such as a heartbeat error), the module changes into the status which has been defined in variable *communication_error*.

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Default [Dec] | Data type | R/W |
|---|---|---|---|---|---|---|
| **1029** | 0 | *no_of_error_classes* | 1 | 1 | byte | ro |
| | 1 | *communication_error* | 0, 1, 2 | 0 | byte | rw |

Meaning of variables:

*no_of_error_classes* Number of error classes (here always ‘1’)

*communication_error* 0 - pre-operational (only, if current status = operational)
1 - no state change
2 - stopped

### 2.3.17 Object Transmit PDO Communication Parameter $1800_h$

Via this object the features of a transmit PDO are defined.

| **INDEX** | **$1800_h$** |
|---|---|
| Name | *transmit PDO parameter* |
| Data Type | PDOCommPar |

| Index | Sub-index [Dec] | Description | Value range [Hex] | Default [Hex] | Data type | R/W |
|---|---|---|---|---|---|---|
| **1800** | $0_h$ | number of entries | 5 | $5_h$ | unsigned 8 | const. |
| | $1_h$ | COB-ID used by PDO | 0...FFFFFFFF | $180_h$ + Node-ID | unsigned 32 | ro |
| | $2_h$ | transmission type | 0...FF | $255_d$ | unsigned 8 | rw |
| | $5_h$ | event timer | 0...FFFFFFFF | 0 | unsigned 16 | rw |

Value range in accordance with DS301

### 2.3.18 Transmit PDO Mapping Parameter $1A00_h$

Via object 'Transmit PDO Mapping Parameter $1A00_h$' the assignment of transmit data to Tx-PDOs can be changed.

| **INDEX** | **$1A00_h$** |
|---|---|
| Name | *transmit PDO mapping* |
| Data Type | PDO Mapping |

The following table represents the assignment of transmit PDO-mapping parameters for default configuration:

| Index | Sub-index [Dec] | Description | Value range [Hex] | Default [Hex] | Data type | R/W |
|---|---|---|---|---|---|---|
| **1A00** | $0_h$ | number of entries | 4 | $4_h$ | unsigned 8 | ro |
| | $1_h$ | read_analog_Input 16 Bit, sub-index 1, A_In 1 | 0...FFFFFFFF | $64010110_h$ | unsigned 32 | ro |
| | $2_h$ | read_analog_Input 16 Bit, sub-index 2, A_In 2 | 0...FFFFFFFF | $64010210_h$ | unsigned 32 | ro |
| | $3_h$ | read_analog_Input 16 Bit, sub-index 3, A_In 3 | 0...FFFFFFFF | $64010310_h$ | unsigned 32 | ro |
| | $4_h$ | read_analog_Input 16 Bit, sub-index 4, A_In 4 | 0...FFFFFFFF | $64010410_h$ | unsigned 32 | ro |

Value range in accordance with DS301

## 2.4 Transmission Types Supported in Accordance with DS301

| *Transmission Type* | PDO transmission | | | | | Supported by CAN-CBM-AI410 |
|---|---|---|---|---|---|---|
| | cyclic | acyclic | synchronous | asynchronous | RTR only | |
| 0 | - | X | X | - | - | Yes |
| 1...240 | X | | X | - | - | Yes |
| 241...251 | reserved | | | | | No |
| 252 | - | - | X | - | X | Yes |
| 253 | - | - | - | X | X | Yes |
| 254 | - | - | - | X | - | Yes |
| 255 | - | - | - | X | - | Yes |

## 2.5 Node Guarding / Life Guarding

If the CANopen modules are to monitor each other (especially to detect connection failures) either the heartbeat function (see object $1017_h$) or the node/life guarding can be used. With the CAN-CBM-AI410-module the monitoring functions exclude one another and cannot be used simultaneously, therefore.
In contrast to the node guarding/life guarding the heartbeat function does not require RTR-frames.

In CANopen terminology node guarding is the monitoring of the NMT-slave module (here CAN-CBM-AI410) by the NMT-master, life guarding on the other hand occurs on the NMT-slave and monitors the NMT-master.

Node guarding:

The NMT-master cyclically requests a special telegram from its NMT slave (here CAN-CBM-AI410) via an RTR-message (CAN-request). If the response by the slave does not correspond to the response expected, or if the response does not come within the *guard time*, the NMT-master recognizes an error and reacts accordingly. The response telegram contains the module status and a toggle bit.

Life guarding:

The NMT-slaves monitor, whether they are requested by the NMT-master in connection with node guarding. If these requests stop for the period of the *life time*, a local RESET is triggered. The life time is the product from *guard time* (index $100C_h$) and the *life time factor* (index $100D_h$).

# 3. Device Profile Area

## 3.1 Overview

The module only supports the object represented in the table below.

| Index [Hex] | Name | Sub-index [Hex] | Type | Access | Default [Hex] |
|---|---|---|---|---|---|
| 6401 | Read Analog Input 16-bit | $0...4_h$ | Integer 16 | ro | No |
| 6421 | Analog Interrupt Trigger Selection | $0...4_h$ | Unsigned 8 | ro | No |
| 6422 | Analog Input Interrupt Source | $0_h$, $1_h$ | Unsigned 32 | ro | No |
| 6423 | Analog Input Global Interrupt Enable | $0_h$ | Boolean | rw | False |
| 6426 | Analog Input Interrupt Delta Unsigned | $0...4_h$ | Unsigned32 | rw | 0 |

ro - Read Only, rw - Read/Write

### 3.1.1 Read Analog Input 16-bit $6401_h$

| **INDEX** | **$6401_h$** |
|---|---|
| Name | *read analog input* |
| Data Type | Integer 16 |
| Default Value | No |

The object reads the values of analog inputs 1...4.

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Default [Hex] | Data type | R/W |
|---|---|---|---|---|---|---|
| **6401** | 0 | *number of entries* | 4 | - | byte | ro |
| | 1 | *Read_Analog_Input_1* | 0000...7FE0 | - | integer16 | ro |
| | 2 | *Read_Analog_Input_2* | 0000...7FE0 | - | integer16 | ro |
| | 3 | *Read_Analog_Input_3* | 0000...7FE0 | - | integer16 | ro |
| | 4 | *Read_Analog_Input_4* | 0000...7FE0 | - | integer16 | ro |

| Byte | 0 | | | | | | | | 1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Contents | sign | analog value value range: $0000_h$... $7FE0_h$ | | | | | | | | | | 0 | 0 | 0 | 0 | 0 |

The analog values correspond to the respective input voltages in the range from 0...10 V.
$0000_h$ is 0V and the maximum input voltage of 10 V corresponds to $7FE0_h$.

### 3.1.2 Analog Input Interrupt Trigger $6421_h$

By means of this object you can configure the moment an interrupt is to be triggered.

| **INDEX** | **$6421_h$** |
|---|---|
| Name | *interrupt trigger selection* |
| Data Type | unsigned8 |
| Default Value | see table below |

| Index | Sub-index [Dec] | Description | Value range [Hex] | Default [Hex] | Data type | R/W |
|---|---|---|---|---|---|---|
| **6421** | $0_h$ | *number of analog entries* | 4 | 4 | unsigned 8 | ro |
| | $1_h$ | *Analog_Input_1* | 0...FF | 0 | unsigned 8 | rw |
| | $2_h$ | *Analog_Input_ 2* | 0...FF | 0 | unsigned 8 | rw |
| | $3_h$ | *Analog_Input_3* | 0...FF | 0 | unsigned 8 | rw |
| | $4_h$ | *Analog_Input_4* | 0...FF | 0 | unsigned 8 | rw |

| Bit: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Meaning: | reserved for future applications | | | not being supported | | **value has changed for more than delta ($6426_h$)** | not being supported | |

### 3.1.3 Analog Input Interrupt Source $6422_h$

| **INDEX** | **$6422_h$** |
|---|---|
| Name | *analog input interrupt source* |
| Data Type | unsigned 32 |
| Default Value | No |

This object represents the channel which has triggered an interrupt.

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Default [Hex] | Data type | R/W |
|---|---|---|---|---|---|---|
| **6422** | 0 | *no_of_interrupt_source* | 1 | 1 | unsigned 8 | ro |
| | 1 | *interrupt_source* | 0...F | - | unsigned 32 | ro |

Meaning of variables:

*interrupt_source* This variable shows the channel which has triggered an interrupt.

1... interrupt triggered
0... no interrupt

The values for the four channels are transmitted in the LSB:

| Bit: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Contents: | 0 | 0 | 0 | 0 | *channel_4* | *channel_3* | *channel_2* | *channel_1* |

### 3.1.4 Global Interrupt Enable $6423_h$

| **INDEX** | **$6423_h$** |
|---|---|
| Name | *global interrupt enable* |
| Data Type | Boolean |
| Default Value | see table below |

By means of object Analog Input Global Interrupt Enable the interrupt function of the module is enabled or disabled. The values of objects $6421_h$, $6422_h$ and $6426_h$ remain unchanged.
The default value is set in a way that no change of the analog input value triggers an interrupt.

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Default [Hex] | Data type | R/W |
|---|---|---|---|---|---|---|
| **6423** | 0 | *Analog_Input_Global_Interrupt_Enable* | True, False | False | Boolean | rw |

True... Global Interrupt enabled
False... Global Interrupt disabled

### 3.1.5 Analog Input Interrupt Delta $6426_h$

| **INDEX** | **$6426_h$** |
|---|---|
| Name | *analog input interrupt delta unsigned* |
| Data Type | Unsigned32 |
| Default Value | No |

Via this object the deviation of the analog value (delta) can be set which can be evaluated for the triggering of an interrupt. The mode of evaluation is selected via object $6421_h$. The interrupt is enabled via object $6423_h$.

| Index [Hex] | Sub-index [Dec] | Description | Value range [Hex] | Default [Hex] | Data type | R/W |
|---|---|---|---|---|---|---|
| **6426** | 0 | *Number_Analog_Inputs* | 4 | 4 | Unsigned32 | ro |
| | 1 | *Analog_Input_1* | 0...FFFF | 0 | Unsigned32 | rw |
| | 2 | *Analog_Input_2* | 0...FFFF | 0 | Unsigned32 | rw |
| | 3 | *Analog_Input_3* | 0...FFFF | 0 | Unsigned32 | rw |
| | 4 | *Analog_Input_4* | 0...FFFF | 0 | Unsigned32 | rw |

# 4. PDO-Assignment

The PDOs (Process Data Objects) are used to transmit the process data:

| Tx-PDO | Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte 7 | Byte 8 |
|---|---|---|---|---|---|---|---|---|
| 180 + Node-ID | Analog_Input 1 | | Analog_Input 2 | | Analog_Input 3 | | Analog_Input 4 | |

This page has intentionally been left blank.

# 5. Quick Start

## 5.1 Configuration for the Use in a CANopen-Network

For a quick start with the most simple configuration you have to follow the steps below:

1. Wire the CAN (do not forget the terminations!)

2. Set bit rate:
   (Only if another bit rate than the default setting is desired.)
   The default bit rate is 125 kbit/s. The bit rate can be set via the coding switches (SW100, SW101), as described in the hardware manual (chapter: 'Manual Configuration via Coding Switches').

3. Set module number (Node-ID):
   If you desire a module number different from the default-set number, or if the bit rate has been changed, the module number has to be set via the coding switches (SW100, SW101), as described in the hardware manual (chapter Configuration via Coding Switches).
   The module number can be set with values between 1 and 127 ($01...7F_h$).

   Example: Setting module number 1 ($01_h$):
   - switch off power supply
   - turn coding switch SW100 (Low) to '1', SW 101 (High) to '0'
   - switch on power supply, module wakes up in *Preoperational* status

4. Switch on module:
   Apply power.
   If the module number has been changed before, power is already supplied and the module is in *Preoperational* status (see hardware manual chapter: Module-Status LEDs).

5. Start module with:

| CAN-Identifier | Len | Data | |
|---|---|---|---|
| 0 | 2 Byte | 1 | 0 |

6. Request status analog inputs with:

| CAN-Identifier | RTR | Len |
|---|---|---|
| 181 | 1 | 0 |

7. Module responds with:

| CAN-Identifier | Len | Data |
|---|---|---|
| 181 | 8 Byte | depending on status of inputs |

## 5.2 Table of Important Identifiers and Messages for CANopen

| CAN-Identifier [Hex] | Designation | Length | Data [Hex] | Notes |
|---|---|---|---|---|
| 0 | NMT | 2 | 02 xx | CAN-CBM-AI410-module goes into *Stopped* status |
| 0 | NMT | 2 | 01 xx | start (CAN-CBM-AI410-module in *Operational* status) |
| 0 | NMT | 2 | 80 xx | CAN-CBM-AI410-module in *Preoperational* status |
| 0 | NMT | 2 | 81 xx | reset CAN-CBM-AI410-module |
| 0 | NMT | 2 | 82 xx | Reset communication (here same function as '81xx') |
| $80_h$ | NMT | 0 | - | Sync to all |
| $80_h$ + Node-ID | EMCY | 8 | see page 14 | emergency frame from CAN-CBM-AI410 |
| $700_h$ + Node-ID | NMT | 8 Bytes | identifier | NMT error control identifier (see DS301) |
| $580_h$ + Node-ID | SDO | 8 Bytes | parameter | acknowledging the communication parameters of CAN-CBM-AI410-module (Tx) |
| $600_h$ + Node-ID | SDO | 8 Bytes | parameter | transmission of communication parameters to CAN-CBM-AI410-module (Rx) |
| $180_h$ + Node-ID | Tx_PDO_1 | 1 Byte | user data | (Tx/transmit-PDO) |

xx = Node-ID (module number)
Node-ID = $1...7F_h$