# Operating instructions
# RFID read/write head
# with CANopen interface

UK

706404 / 03 08 / 2020

# Contents

# 1 Preliminary note

This document applies to devices of the type "RFID read/write head with CANopen interface" (e.g. art. no.: DTM425). This document is part of the device.

This document is for specialists. These specialists are people who are qualified by their appropriate training and their experience to see risks and to avoid possible hazards that may be caused during operation or maintenance of the device. The document contains information about the correct handling of the device.

Read this document before use to familiarise yourself with operating conditions, installation and operation. Keep this document during the entire duration of use of the device.

Adhere to the safety instructions.

## 1.1 Symbols used

► Instruction
> Reaction, result
[…] Designation of keys, buttons or indications
→ Cross-reference

Important note
Non-compliance may result in malfunction or interference.

Information
Supplementary note

# 2 Safety instructions

## 2.1 General

These instructions are an integral part of the device. They contain texts and figures concerning the correct handling of the device and must be read before installation or use.

Observe the operating instructions. Non-observance of the instructions, operation which is not in accordance with use as prescribed below, wrong installation or incorrect handling can seriously affect the safety of operators and machinery.

## 2.2 Target group

These instructions are intended for authorised persons according to the EMC and low-voltage directives. The device must be installed, connected and put into operation by a qualified electrician.

## 2.3 Electrical connection

Disconnect the unit externally before handling it.

The connection pins may only be supplied with the signals indicated in the technical data and/or on the device label and only the approved accessories of ifm may be connected.

The device does not have an internal CAN terminating resistor. A connection cable without terminating resistor can cause interference on the CAN bus.

► Use 120 Ω terminating resistors or a connection cable with integrated terminating resistor, e.g. article EVC492 .



### 2.4 Tampering with the device

In case of malfunctions or uncertainties please contact the manufacturer. Any tampering with the device can seriously affect the safety of operators and machinery. This is not permitted and leads to the exclusion of any liability and warranty claims.

## 3 General information

### 3.1 CANopen technology

The CANopen communication profile is based on the CAN Application Layer (CAL) specification of the CiA organisation. CANopen is considered as a robust fieldbus with highly flexible configuration options. It is used in many different applications which are based on different application profiles. CANopen comprises a concept to configure and communicate real-time data using synchronous and asynchronous messages. Four message types (objects) are distinguished.

1. Administration messages (layer management, network management and identifier distribution)
2. Service Data Objects (SDO)
3. Process Data Objects (PDO)
4. Predefined Objects (emergency)

For further information please refer to the CiA-CAN specification (CiA 301 - CANopen).

### 3.2 Reference

http://www.can-cia.org

| | |
|---|---|
| CAN Application Layer, DS 201 …207 | CiA |
| LSS profile | DS305 CiA |
| CAN-based communication profile | DS 301 CiA |
| CAN specification version 2.0 A | Robert Bosch GmbH |

# 4 Functions and features

The RFID read/write head is used for reading of and writing to ID tags. The read/write head is configured and data is exchanged via the integrated CANopen interface.

Typical applications are for example the identification of interchangeable tools and attachments on mobile machines.

# 5 Installation

## 5.1 General installation instructions

- Observe the separate mounting instructions.
- When mounting several read/write heads adhere to the minimum distances between the systems.
- The immediate vicinity of powerful HF emission sources such as welding transformers or converters can affect operation of the read/write heads.

## 5.2 Notes on ID tag mounting

- Installation of the ID-Tags in or on metal reduces the read and write distances
- The orientation of the read/write head antenna axis must correspond with the axis of the ID tag coil.

## 5.3 Avoiding interference

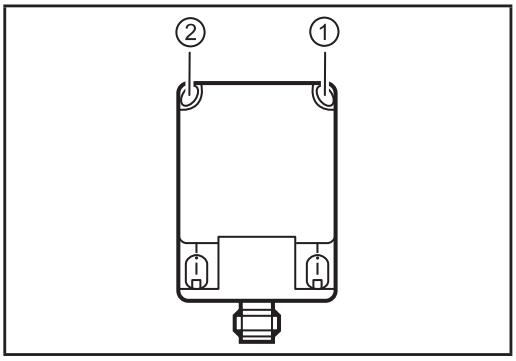
The device generates a modulated electrical field with a frequency of 13.56 MHz. To avoid interference of the data communication no other devices generating interference emission in this frequency band must be operated in its vicinity. Such devices are for example frequency converters and switched-mode power supplies.

# 6 Indicators of the DTM424 / 425 / 428 / 434 / 435

| Operating status | LED red | LED green | LED yellow |
|---|---|---|---|
| preoperational | off | permanently on | off |
| preoperational and ID tag detected | off | flashes alternately with yellow LED (every 1.6 s) | flashes alternately with green LED (every 1.6 s) |
| operational | off | flashes (every 0.4 s) | off |
| operational and ID tag detected | off | off | permanently on |
| configuration error | flashes (every 0.4 s) | LED reacts according to the current operating status | |
| error in the CAN network | flashes (every 1.2 s) | | |
| CAN bus off | permanently on | off | off |
| LSS service active | flashes | off | off |
| hardware error detected in the device | off | off | flashes |

UK

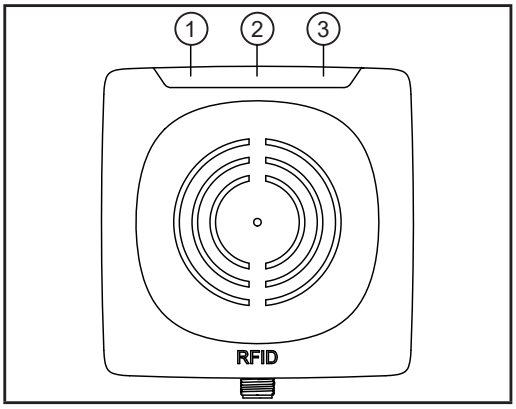
# 7 Indicators of the DTC510




1: green (operating status) / red (error)
2: yellow (ID tag)

| Operating status | LED red | LED green | LED yellow |
|---|---|---|---|
| preoperational | off | on | off; on if an ID tag has been detected in the reading field |
| operational | off | flashes (2.5 Hz) | off; on if an ID tag has been detected in the reading field |
| configuration error | flashes alternately with green LED (2.5 Hz) | flashes alternately with red LED (2.5 Hz) | off; on if an ID tag has been detected in the reading field |
| error in the CAN network | flashes alternately with green LED (0.8 Hz) | flashes alternately with red LED (0.8 Hz) | off; on if an ID tag has been detected in the reading field |
| CAN bus off | on | off | off; on if an ID tag has been detected in the reading field |
| LSS service active | flickers irregularly | off | off; on if an ID tag has been detected in the reading field |
| hardware error detected in the device | off | off | flickers irregularly |

# 8 Indicators of the DTC600




1: LED operating status (green)
2: 4 LEDs signal strength (yellow)
3: LED CANopen status (green/red)

| LED operating status (green) | Description |
|---|---|
| off | unit not operational (without power supply) |
| on | unit operational (high-frequency field is generated) |
| flashes | unit in standby (high-frequency field deactivated) |

| 4 LEDs signal strength (yellow) | Description |
|---|---|
| off | no ID tag in the field |
| up to 4 LEDs on | depending on the signal strength of the ID tag in the field, up to 4 LEDs are on |
| flashing once | successful read or write access to ID tag |
| flashing fast several times | unsuccessful read or write access to ID tag |

| LED CANopen status (green/red) | Description |
|---|---|
| green permanently on | preoperational status |
| green flashing (2.5 Hz) | operational status |
| flashing green or red alternately (2.5 Hz) | configuration error |
| flashing green or red alternately (0.8 Hz) | error in the CAN network |
| red permanently on | CAN bus off |
| red flickers irregularly | LSS service on |

# 9 CANopen interface

The RFID read/write head has a standardised CANopen interface according to CiA DS-301. All measured values and parameters can be accessed via the object directory (OD). The individual configuration can be saved in the internal permanent memory.

## 9.1 CANopen functions

The following CANopen functions are available:



- 64 transmit and receive process data objects (TPDO1..64, RPDO1..64) in two possible operating modes:
  - individual check via a remote transmit request telegram (RTR)
  - event-controlled transmission
- Error messages per emergency object (EMCY) with support of the:
  - general error register
  - manufacturer-specific status register
  - error list
- Heartbeat monitoring mechanism
- Status and error indication via LED
- In addition to the CiA DS-301 functionality there are more manufacturer and profile-specific characteristics:
  - setting of the node ID and the bit rate via object directory entry (SDO)
  - configuration and reading/writing of operational data via service data objects (SDO)
- Support of the layer settings service (LSS)
- Support of synchronous process data transmission (SYNC)

## 9.2 Change the node ID and bit rate

The device supports several options how to change the node ID and the bit rate.

The device is delivered with the node ID 32 and a bit rate of 125 Kbits/s.

Each node ID must only be assigned once in the CANopen network. If a node ID is assigned several times, malfunction in the CANopen network will result.

### 9.2.1 Change the node ID and bit rate in the object directory

The node ID is entered in the object directory in the objects 0x20F0 and 0x20F1. If the two values are identical, the setting is stored and is active after a software reset of the device. Values between 1 and 127 may be used as node ID.

The bit rate is entered in the objects 0x20F2 and 0x20F3. If the two values are identical, the setting is stored and is active after a software reset of the device. The following values may be used as bit rate:

| Value | Bit rate |
|---|---|
| 0 | 1000 kBits/s |
| 1 | 800 kBits/s |
| 2 | 500 kBits/s |
| 3 | 250 kBits/s |
| 4 | 125 kBits/s |
| 5 | 100 kBits/s |
| 6 | 50 kBits/s |
| 7 | 20 kBits/s |

If a master is used in the CANopen network for central storage of parameters, the changed values for node ID (0x20F0 and 0x20F1) and bit rate (0x20F2 and 0x20F3) must be additionally entered in the master.

Otherwise the values will be reset during each start of the CANopen network.

### 9.2.2 Set the note ID and bit rate via LSS

Using the layer setting service (LSS) an LSS master can change the node ID and bit rate of the device (LSS slave) via the CAN bus. The LSS master sets all LSS slaves to a configuration mode. Each LSS slave can be unambiguously identified via the device data (vendor ID, product code, revision number and serial number).

To change the bit rate the LSS master transfers the new bit rate in the configuration mode with the service "Configure timing bit". The LSS slave replies to the LSS master if the new bit rate is supported. Then the LSS master transmits the time "switch_delay" via the service "Activate bit timing” after which the new bit rate should be activated. After activation the LSS master switches the LSS slave again to the operating mode.

To change the node ID the LSS master transfers the new node ID in the configuration mode. The LSS slave replies to the master if the new node ID is valid. After changing the node ID the LSS master switches the LSS slave again to the operating mode.

The new bit rate and node ID become active after a software reset of the LSS slave.

## 9.3 Set-up

The CANopen standard CiA301 defines three possible operating states:

**Pre-Operational**
In the pre-operational state no PDO messages (process data) can be transmitted. The pre-operational state is used to set the sensor parameters or as standby mode.

During booting in the pre-operational mode, the device reports the bootUP message "0x700+Node-ID" to the CAN bus.



**Operational**
In the operational state all communication services are carried out. The operational state is used to exchange the process data while in operation.

**Stopped**
In the stopped state only NMT messages (network management) are possible. This allows almost complete separation of redundant or faulty sensors from the bus.

The master or network manager can request the sensor via NMT messages to change the state accordingly.

## 9.4 Use with 32-bit data types

CANopen defines data types with a maximum size of 64 bits (8 bytes). By means of the data type, the user data of ID tags is transmitted efficiently via the CANopen interface. The data type is also used for the default setting of the device and the EDS file.

However, some controllers can only process data types with a maximum width of 32 bits (4 bytes). In order to support all types of controllers, the device offers alternative data objects whose data types are restricted to max. 32 bits. These data objects are marked by the addition “32 bits” in these instructions. Additonally, an EDS file is supplied for use of the data types that is read by the controller software.

By default, the device uses 64-bit data types (e.g. for the preconfigured PDOs). The setting must be adapted to the use of 32-bit data types. The setting can be changed via the controller software, by reading the corresponding EDS file.

## 9.5 Communication types of the process data object (PDO)

The TPDO can be checked at any time by transmitting a remote transmission request telegram (RTR). Otherwise the TPDOs are sent automatically as soon as their value changes (event-driven).

As an option, the CANOpen service "SYNC" can be used (see CiA 301, 7.2.5 Synchronization object (SYNC)). For the synchronised transmission CANopen provides the SYNC object at which the TPDOs are transmitted after every "nth" reception of a SYNC telegram.

A total of 64 TPDOs and 64 RPDOs is available; on delivery only the first 4 of each are active. If the configuration of the CANopen network allows it, the remaining process data objects can also be activated.

The process data is assigned to the linear address range in the standard settings of the ID tag. The TPDO1 maps e.g. the first 8 bytes of the user data memory of the ID tag.

Reading of the memory and transmission of the data via TPDO is effected automatically as soon as a new ID tag is detected.

Writing of the data to the ID tag is effected in the same way by write access to the respective RPDO.

Data transfer per process data object is only possible in the "Operational" operating mode (→ 9.3 Set-up).

The preset TPDOs and RPDOs are allocated 64-bit data objects. For use of 32-bit controllers, the settings of the PDOs must be adapted (→ 9.4 Use with 32-bit data types).

## 9.6 Object directory (OD)

| Index | Subindex | Name (object) | Type | Access | Default value | PDO mapping capability | Save object value |
|---|---|---|---|---|---|---|---|
| **CANopen communication (CiA 301)** | | | | | | | |
| 0x1000 | 0x00 | Device type | u32 | ro | 0x00000000 | - | - |
| 0x1001 | 0x00 | Error register | u8 | ro | 0x00 | - | - |
| 0x1003 | 0x01<br>0x02 | Predefined error field | 032 | ro | 0x00000000 | | |
| 0x1005 | 0x00 | COB ID SYNC | u32 | rw | 0x00000000 | - | yes |
| 0x1008 | 0x00 | Manufacturer device name | vSTR | ro | article no. of the device | - | - |
| 0x1009 | 0x00 | Manufacturer hardware version | vSTR | ro | current hardware version | - | - |
| 0x100A | 0x00 | Manufacturer software version | vSTR | ro | current software version | - | - |
| 0x1010 | 0x01 | Save parameters<br>(save device parameters in non-volatile memory) | u32 | rw | 0x00000000 | - | - |
| 0x1011 | 0x01 | Load default communication parameters | u32 | rw | 0x00000000 | - | - |
| 0x1014 | 0x00 | COB ID EMCY<br>(COB ID emergency message) | u32 | rw | Node ID+ 0x80 | - | |
| 0x1015 | 0x00 | Inhibit time EMCY<br>(inhibit time between EMCY messages) | u16 | rw | 0x0000 | - | yes |
| 0x1017 | 0x00 | Producer heartbeat time<br>(time difference between sent heartbeats in ms) | u16 | rw | 0x0000 | - | yes |
| 0x1018 | 0x01 | Vendor ID | u32 | ro | 0x0069666D | - | - |
| | 0x02 | Product code | u32 | ro | product code of the device version | - | - |
| | 0x03 | Revision number | u32 | ro | main revision and current software version | - | - |
| | 0x04 | Serial number | u32 | ro | serial number of the device | - | - |

UK

| Index | Subindex | Name (object) | Type | Access | Default value | PDO mapping capability | Save object value |
|---|---|---|---|---|---|---|---|
| 0x1200 | 0x01 | COB ID client to server | u32 | ro | Node ID+ 0x600 | - | - |
| | 0x02 | COB ID client to server | u32 | ro | Node ID+ 0x580 | - | - |
| 0x1400-0x143F | 0x01 | RPDO parameter: COB ID | u32 | rw | (→ 9.10.2) | - | yes |
| | 0x02 | RPDO parameter: transmission type | u8 | ro | 0xFF | - | yes |
| 0x1600-0x163F | 0x01-0x08 | RPDO mapping | u32 | rw | (→ 9.10.2) | - | yes |
| 0x1800-0x183F | 0x01 | TPDO parameter: COB ID | u32 | rw | (→ 9.10.1) | - | yes |
| | 0x02 | TPDO parameter: transmission type | u8 | rw | 0xFF | - | yes |
| | 0x03 | TPDO parameter: inhibit time | u16 | rw | 0x00 | - | yes |
| 0x1A00-0x1A3F | 0x01-0x08 | TPDO mapping | u32 | rw | (→ 9.10.1) | - | yes |
| **Bus configuration** | | | | | | | |
| 0x20F0 | 0x00 | NODE ID setting A (node ID for CANopen communication) | u8 | rw | 32 | - | auto-save |
| 0x20F1 | 0x00 | NODE ID setting B (node ID for CANopen communication) | u8 | rw | 32 | - | auto-save |
| 0x20F2 | 0x00 | Bit rate setting A (CAN bus bit rate) | u8 | rw | 4 | - | auto-save |
| 0x20F3 | 0x00 | Bit rate setting B (CAN bus bit rate) | u8 | rw | 4 | - | auto-save |
| **Status and control of the reader** | | | | | | | |
| 0x2150 | 0x00 | Device status (device status flags) | u32 | ro | | yes | - |
| 0x2151 | 0x00 | Antenna active (enable HF front end of the device) | bool | rw | 1 | - | yes |
| 0x2160 | 0x01-0xFE | Definition ID tag type (name of supported tags) | dom | ro | (→ 9.13) | - | - |
| 0x2161 | 0x00 | Selection ID tag type (value selects ID tag type defined in 0x2160) | u8 | rw | 2 | - | yes |

| Index | Subindex | Name (object) | Type | Access | Default value | PDO mapping capability | Save object value |
|---|---|---|---|---|---|---|---|
| 0x2162 | 0x00 | RSSI | u8 | ro | - | yes | - |
| **ID tag information** | | | | | | | |
| 0x2180 | 0x00 | Current UID<br>(UID of the ID tag in the read range, PDO mappable) | u64 | ro | 0x00000000 00000000 | yes | - |
| 0x2181 | 0x00 | Current DSFID<br>(DSFID of the ID tag in the read range, PDO mappable) | u8 | ro | 0x00 | yes | - |
| 0x2182 | 0x01 | ID tag information: UID | u64 | ro | 0x00000000 00000000 | - | - |
| | 0x02 | ID tag information: DSFID | u8 | ro | 0x00 | - | - |
| | 0x03 | ID tag information: AFI | u8 | ro | 0x00 | - | - |
| | 0x04 | ID tag information: memory size | u32 | ro | 0x00000000 | - | - |
| | 0x05 | ID tag information: IC reference | u8 | ro | 0x00 | - | - |
| | 0x06 | ID tag information: ID tag type<br>(detected ID tag type, defined in 0x2160) | u8 | ro | 0x00 | - | - |
| 0x2190 | 0x00 | Current UID 4 upper bytes (32 bits)<br>(UID of the ID tag in the read range, PDO mappable) | u32 | ro | 0x00000000 00000000 | yes | - |
| 0x2191 | 0x00 | Current UID 4 lower bytes (32 bits)<br>(UID of the ID tag in the read range, PDO mappable) | u32 | ro | 0x00000000 00000000 | yes | - |
| **Read mappable data** | | | | | | | |
| 0x2200 | 0x01-0x40 | Address read start point<br>(start of the address range on the ID tag to be read) | u16 | rw | (→ 9.10.2) | - | yes |
| 0x2201 | 0x01-0x40 | Read length<br>(length of the memory range on the ID tag to be read; max. 8 bytes) | u8 | rw | (→ 9.10.2) | - | yes |

| Index | Subindex | Name (object) | Type | Access | Default value | PDO mapping capability | Save object value |
|---|---|---|---|---|---|---|---|
| 0x220A | 0x01-0x40 | ID tag data<br>(8 bytes of ID tag data, updated when new ID tag enters the read area) | u64 | ro | | yes | - |
| 0x220B | 0x01-0x40 | ID tag data (32 bits)<br>(4 bytes of ID tag data, updated when new ID tag enters the read area) | u32 | ro | | yes | - |
| **Read data range** | | | | | | | |
| 0x2280 | 0x00 | Address read start point<br>(start of the address range on the ID tag to be read) | u16 | rw | 0x0000 | - | yes |
| 0x2281 | 0x00 | Read length<br>(length of the memory area on the ID tag to be read) | u16 | rw | 0x0000 | - | yes |
| 0x2282 | 0x00 | ID tag data<br>(requested data from the ID tag as configured in objects 0x2280 and 0x2281) | dom | ro | | - | - |
| **Write data mappable** | | | | | | | |
| 0x2300 | 0x01-0x40 | Address write start point<br>(start of the address range on the ID tag to be written) | u16 | rw | (→ 9.10.1) | - | yes |
| 0x2301 | 0x01-0x40 | Write length<br>(length of the memory range on the ID tag to be written, max. 8 bytes) | u8 | rw | (→ 9.10.1) | - | yes |
| 0x2302 | 0x01-0x40 | Auto-write<br>(automatic write access if a new ID tag is detected) | bool | rw | 0 | - | yes |
| 0x230A | 0x01-0x40 | ID tag data<br>(8 bytes of ID tag data) | u64 | rw | | yes | - |
| 0x230B | 0x01-0x40 | ID tag data (32 bits)<br>(4 bytes of ID tag data) | u32 | rw | | yes | - |
| 0x231E | 0x00 | Write trigger (32 bits)<br>upper PDOs | u32 | rw | 0x00000000<br>00000000 | yes | - |
| 0x231F | 0x00 | Write trigger (32 bits)<br>lower PDOs | u32 | rw | 0x00000000<br>00000000 | yes | - |

| Index | Subindex | Name (object) | Type | Access | Default value | PDO mapping capability | Save object value |
|---|---|---|---|---|---|---|---|
| **Write data range** | | | | | | | |
| 0x2380 | 0x00 | Address write start point<br>(start of the address range on the ID tag to be written) | u16 | rw | 0x0000 | - | yes |
| 0x2381 | 0x00 | Write length<br>(length of the memory area on the ID tag to be written) | u16 | rw | 0x0000 | - | yes |
| 0x2382 | 0x00 | ID tag data<br>(data to be written to the ID tag as configured in objects 0x2380 and 0x2381) | dom | wo | | - | - |
| **Lock data range** | | | | | | | |
| 0x2480 | 0x00 | Address lock start point<br>(start of the address range on the ID tag to be locked, must correspond to the ID tag areas) | u16 | rw | 0x0000 | - | yes |
| 0x2481 | 0x00 | Lock length<br>(length of the memory range on the ID tag to be locked, must correspond to the ID tag ranges) | u16 | rw | 0x0000 | - | yes |
| 0x2482 | 0x00 | Tag lock trigger<br>(trigger for locking data on the ID tag as configured in objects 0x2480 and 0x2481) | bool | wo | | - | - |
| **UID filter** | | | | | | | |
| 0x4603 | 0x00 | UID filter depth | s8 | rw | 0x00 | - | yes |
| 0x4605 | 0x00 | Zero ID filter depth | s8 | rw | 0x02 | - | yes |



## 9.7 Error messages

The device supports a number of emergency messages that are sent in the event of a communication, hardware or RFID error. If one of these errors occurs, the error register (OD index 0x1001) and the pre-defined error field (OD index 0x1003) are updated.

The COB ID of the emergency message can be changed in the object "COB ID EMCY" (OD index 0x1014). By setting bit 31 in this object the emergency messages are deactivated.

The disable time between two emergency messages can be defined via the object 0x1015. The indication is made in steps of 100 µs.

The COB ID of the emergency messages is preset to 0x80 + node ID.

| Emergency error code | Error register (0x1001) | Manufacturer error code | Manufacturer error name | Emergency description |
|---|---|---|---|---|
| 0x8210 | 0x11 | | | protocol - PDO not processed due to length error |
| 0x8130 | 0x01 | | | monitoring - node guarding or heartbeat error |
| 0x8100 | 0x11 | | | monitoring - general communication error, send in case of bus off |
| 0x5000 | 0x81 | 0x01 | | device hardware error (antenna error) |
| 0x4200 | 0x09 | 0x02 | | device temperature too high |
| 0xFF00 | 0x81 | 0x01 | RX: ISO_ COMMAND_ ERROR_NO_ RESPONSE | ID tag did not answer, maybe ID tag is no longer in the field |
| 0xFF00 | 0x81 | 0x02 | RX: ISO_ COMMAND_ ERROR_RX_ ERROR | error while receiving the answer from the ID tag (CRC error, framing error, collision, etc.) |
| 0xFF01 | 0x81 | 0x01 | TX: ISO_ COMMAND_ ERROR_NO_ RESPONSE | ID tag did not answer, maybe ID tag is no longer in the field |
| 0xFF01 | 0x81 | 0x02 | TX: ISO_ COMMAND_ ERROR_RX_ ERROR | error while receiving the answer from the ID tag (CRC error, framing error, collision, etc.) |
| 0xFF02 | 0x81 | 0x01 | ISO_TAG_ ERROR_ COMMAND_NOT_ SPECIFIED | The specified command is not supported. Example: command code error |
| 0xFF02 | 0x81 | 0x02 | ISO_TAG_ ERROR_ COMMAND_ SYNTAX | Cannot recognise the command. The number of blocks is too high. Example: format error |
| 0xFF02 | 0x81 | 0x03 | ISO_TAG_ ERROR_ OPTION_NOT_ SUPPORTED | indicated options are not supported |
| 0xFF02 | 0x81 | 0x0F | ISO_TAG_ ERROR_OTHER | other errors |

| Emergency error code | Error register (0x1001) | Manufacturer error code | Manufacturer error name | Emergency description |
|---|---|---|---|---|
| 0xFF02 | 0x81 | 0x10 | ISO_TAG_ ERROR_BLOCK_ NOT_USABLE | The specified block cannot be used (or was not found). |
| 0xFF02 | 0x81 | 0x11 | ISO_TAG_ ERROR_BLOCK_ ALREADY_ BLOCKED | The specified block has already been locked and cannot be locked again |
| 0xFF02 | 0x81 | 0x12 | ISO_TAG_ ERROR_ BLOCK_NOT_ UPDATEABLE | The specified block has already been locked and its contents cannot be updated |
| 0xFF02 | 0x81 | 0x13 | ISO_TAG_ ERROR_BLOCK_ WRITE_VERIFY | The specified block could not be programmed normally (a write verify error occurred). |
| 0xFF02 | 0x81 | 0x14 | ISO_TAG_ ERROR_BLOCK_ LOCK_VERIFY | The specified block could not be locked normally (a lock verify error occurred) |
| 0xFF03 | 0x81 | 0x00 | STATUS_ BUFFER_OVERFL | internal buffer overflow |



## 9.8 Monitoring activity via heartbeat

By means of the heartbeat function the activity of a device in the CANopen network can be monitored by the master. The device regularly sends a heartbeat message containing the device status.

The heartbeat function is activated by entering a value greater than "0" into the heartbeat interval time object (OD index 0x1017). The value indicates the time between two heartbeat signals in milliseconds. The heartbeat function is deactivated with the value "0".

## 9.9 Change objects

Changes of the objects in the object directory are applied at once. The changes will get lost by a reset. To prevent this the objects have to be saved in the internal permanent memory (flash). All objects marked in the object directory as "Save object value: yes" are permanently stored in the device flash. By writing the command "save" (65766173h) to save the objects (OD index 1010h/01h) all current objects of the object directory are transferred to the flash memory.

The objects can be reset to factory setting by writing the command "load" (64616F6Ch) to the OD index 1011h/01h. After a reset the changes are applied.

Depending on the architecture of the CANopen network the objects can also be stored centrally in a CANopen master. In this case the objects are transferred to the device when the system is started and the locally stored values are overwritten.

Special features of the object node ID (OD index 0x20F0 and 0x20F1) and the bit rate (OD index 0x20F2 and 0x20F3):

- Changes of the objects are only applied after a reset (→ 9.2 Change the node ID and bit rate).
- The objects cannot be transferred to the flash via the OD index 1010h/01h.
- The objects cannot be reset to the factory setting via the OD index 1011h/01h.

## 9.10 Process data objects

64 transmit and receive process data objects each are available. On delivery 4 process data objects are active.

### 9.10.1 Transmit process data objects (TPDO)

The table below contains the transmit process data objects (TPDO) on delivery.

| | **Settings for PDO mapping** | **Object directory** | | | **ID tag memory** | |
|---|---|---|---|---|---|---|
| **TPDO** | **COB** | **Mapped object index** | **Mapped object subindex** | **Mapped object length** | **Address read start point** | **Read length** |
| 1 | Node ID + 0x0180 | 0x2150 | 0x00 | 0x20 | device status | |
| 2 | Node ID + 0x0280 | 0x220A | 0x01 | 0x40 | 0x00000000 | 0x08 |
| 3 | Node ID + 0x0380 | 0x220A | 0x02 | 0x40 | 0x00000008 | 0x08 |
| 4 | Node ID + 0x0480 | 0x220A | 0x03 | 0x40 | 0x00000010 | 0x08 |
| 5 | 0 (deactivated) | 0x220A | 0x04 | 0x40 | 0x00000018 | 0x08 |
| | | | | | | |
| 64 | 0 (deactivated) | 0x220A | 0x3F | 0x04 | 0x000001F0 | 0x08 |

The preset TPDOs and RPDOs are allocated 64-bit data objects. For use of 32-bit controllers, the settings of the TPDOs and RPDOs must be adapted (→ 9.4 Use with 32-bit data types).

### 9.10.2 Receive process data objects (RPDO)

The table below contains the receive process data objects (RPDO) on delivery.

| | Settings for PDO mapping | Object directory | | | ID tag memory | |
|---|---|---|---|---|---|---|
| **RPDO** | **COB** | **Mapped object index** | **Mapped object subindex** | **Mapped object length** | **Address write start point** | **Write length** |
| 1 | Node ID + 0x0200 | 0x230F | 0x00 | 0x40 | write trigger | |
| 2 | Node ID + 0x0300 | 0x230A | 0x01 | 0x40 | 0x00000000 | 0x08 |
| 3 | Node ID + 0x0400 | 0x230A | 0x02 | 0x40 | 0x00000008 | 0x08 |
| 4 | Node ID + 0x0500 | 0x230A | 0x03 | 0x40 | 0x00000010 | 0x08 |
| 5 | 0 (deactivated) | 0x230A | 0x04 | 0x40 | 0x00000018 | 0x08 |
| | | | | | | |
| 64 | 0 (deactivated) | 0x230A | 0x3F | 0x04 | 0x000001F8 | 0x08 |

The preset TPDOs and RPDOs are allocated 64-bit data objects. For use of 32-bit controllers, the settings of the TPDOs and RPDOs must be adapted (→ 9.4 Use with 32-bit data types).

## 9.11 Device status

The current device status is represented in the object "Device status" (OD index 0x2150, subindex 0x00). On delivery the object is assigned to TPDO1.

| **Bit** | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|---|---|---|---|---|---|---|---|---|
| **Status** | tag_err | | | | | | | |
| **Default value** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| **Bit** | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|
| **Status** | write_err | | | | | | | |
| **Default value** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| **Bit** | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|---|
| **Status** | read_err | | | | | | | |
| **Default value** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| **Bit** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Status** | r | r | buf_ovfl | fr_err | busy | present | ant | pow |
| **Default value** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

| **Status** | **Value** | **Description** | **EMCY message** |
|---|---|---|---|
| pow | 1 | power enabled (always 1) | |
| ant | 0 | antenna deactivated | |
| | 1 | antenna enabled | |
| present | 0 | no ID tag present | |
| | 1 | ID tag present | |
| busy | 0 | idle | |
| | 1 | read or write access active | |
| fr_err | 0 | front end OK | |
| | 1 | front end error detected (hardware problem) | yes |
| buf_ovfl | 0 | buffer OK | |
| | 1 | buffer overflow detected | yes |
| read_err | | error of last read operation | yes |
| write_err | | error of last write operation | yes |
| tag_err | | error message from ID-Tag for last operation | yes |

| Read error codes (updated after each read access of the ID tag) | | |
|---|---|---|
| 0x00 | ISO_COMMAND_ERROR_NO_ERROR | no error, command successfully executed |
| 0x01 | ISO_COMMAND_ERROR_NO_ RESPONSE | ID tag did not answer, maybe ID tag is no longer in the field |
| 0x02 | ISO_COMMAND_ERROR_RX_ERROR | error while receiving the answer from the ID tag (CRC error, framing error, collision, etc.) |

| Write error codes (updated after each write access of the ID tag) | | |
|---|---|---|
| 0x00 | ISO_COMMAND_ERROR_NO_ERROR | no error, command successfully executed |
| 0x01 | ISO_COMMAND_ERROR_NO_ RESPONSE | ID tag did not answer, maybe ID tag is no longer in the field |
| 0x02 | ISO_COMMAND_ERROR_RX_ERROR | error while receiving the answer from the ID tag (CRC error, framing error, collision, etc.) |

| ID tag error codes (updated after read or write access of the ID tag) | | |
|---|---|---|
| 0x00 | ISO_TAG_ERROR_NO_ERROR | no error from the ID tag |
| 0x01 | ISO_TAG_ERROR_COMMAND_NOT_ SPECIFIED | The specified command is not supported. Example: command code error |
| 0x02 | ISO_TAG_ERROR_COMMAND_SYNTAX | Cannot recognise the command. The number of blocks is too high. Example: format error |
| 0x03 | ISO_TAG_ERROR_OPTION_NOT_ SUPPORTED | indicated options are not supported |
| 0x0F | ISO_TAG_ERROR_OTHER | other errors |
| 0x10 | ISO_TAG_ERROR_BLOCK_NOT_ USABLE | The specified block cannot be used (or was not found). |
| 0x11 | ISO_TAG_ERROR_BLOCK_ALREADY_ BLOCKED | The specified block has already been locked and cannot be locked again |
| 0x12 | ISO_TAG_ERROR_BLOCK_NOT_ UPDATEABLE | The specified block has already been locked and its contents cannot be updated |
| 0x13 | ISO_TAG_ERROR_BLOCK_WRITE_ VERIFY | The specified block could not be programmed normally (a write verify error occurred). |
| 0x14 | ISO_TAG_ERROR_BLOCK_LOCK_ VERIFY | The specified block could not be locked normally (a lock verify error occurred) |



## 9.12 Deactivate antenna

The antenna in the device can be deactivated if the value 0 is written to the object "Antenna active" (OD index 0x2151). In this case no tag is detected any more since the magnetic field of the device is no longer active.

The antenna is reactivated with the value 1. With the object "Antenna active" it is possible to prevent interference between two devices placed next to each other by alternately deactivating the antennas of the two devices.

## 9.13 Select the ID tag type

The device is compatible with several ID tag types to ISO15693. Depending on the size of the user data memory and manufacturer, the ID tags differ in the access to data. Therefore, the device must know which type of ID tag is used in the system.

In object 0x2161, the ID tag type used in the RFID system can be selected. The available ID tag types can be read in the object 0x2180, subindex 0x01-0xFE.

| ID tag type | Name | Block size [byte] | Number of blocks |
|---|---|---|---|
| 1 | user-defined | ? | ? |
| 2 | I-Code SLI | 4 | 28 |
| 3 | I-Code SLI-S | 4 | 40 |
| 4 | I-Code SLI-L | 4 | 8 |
| 5 | F-MEM 2k | 8 | 250 |
| 6 | F-MEM 232b | 4 | 58 |
| 7 | F-MEM 8k | 32 | 256 |
| 8 | TI_32b | 4 | 8 |
| 9 | TI_256b | 4 | 64 |
| 10 | ST_128b | 4 | 32 |
| 11 | ST_256b | 4 | 64 |
| 12 | ST_8k | 4 | 2048 |
| 13 | I-Code SLIX2 | 4 | 79 |

Via the object 0x2182 0x06 it is possible to poll the ID tag type read by the device. First of all, the detected ID tag type must be read from the object 0x2182 subindex 0x06 and this value must be entered in the object 0x2161.

Of special importance is ID tag type 1: The parameters "Block size" and "Number of blocks" are determined by the device. If the parameters do not match the known ID tag types, type 1 "User defined" is used.

Recognition of the ID tag type is not supported by all ID tags.

The set ID tag type is only permanently saved in the device if the object "Save parameter" is used (→ 9.9 Change objects).

ID tag type 2 is preset.

## 9.14 Read information of an ID tag

The information of an ID tag can be read via the objects 0x2180 to 0x2182. To do so, the ID tag has to be within the detection range of the device.

The objects 0x2180 and 0x2182 are only valid as long as the ID tag is detected. If there is no ID tag within the range, the values of the objects are reset to 0.

The value of the object 0x2182 can be read from the ID tag on request.

Reading of information is not supported by each ID tag type.



## 9.15 RSSI value

The RSSI value (Received Signal Strength, OD index 0x2162) informs about the strength of the received signal that is emitted by the ID tag in front of the device:

0: no ID tag detected

1: minimum signal strength

8: maximum signal strength

The maximum signal strength is only reached with certain device / ID tag combinations.

The signal strength depends on the distance between the ID tag and the active face of the device.

Position changes in the environment, e.g. of metallic objects, can influence the signal strength.

On DTC600, the number of the lit yellow LEDs signal strength does not correspond to the RSSI value.

## 9.16 ID tag detection filter

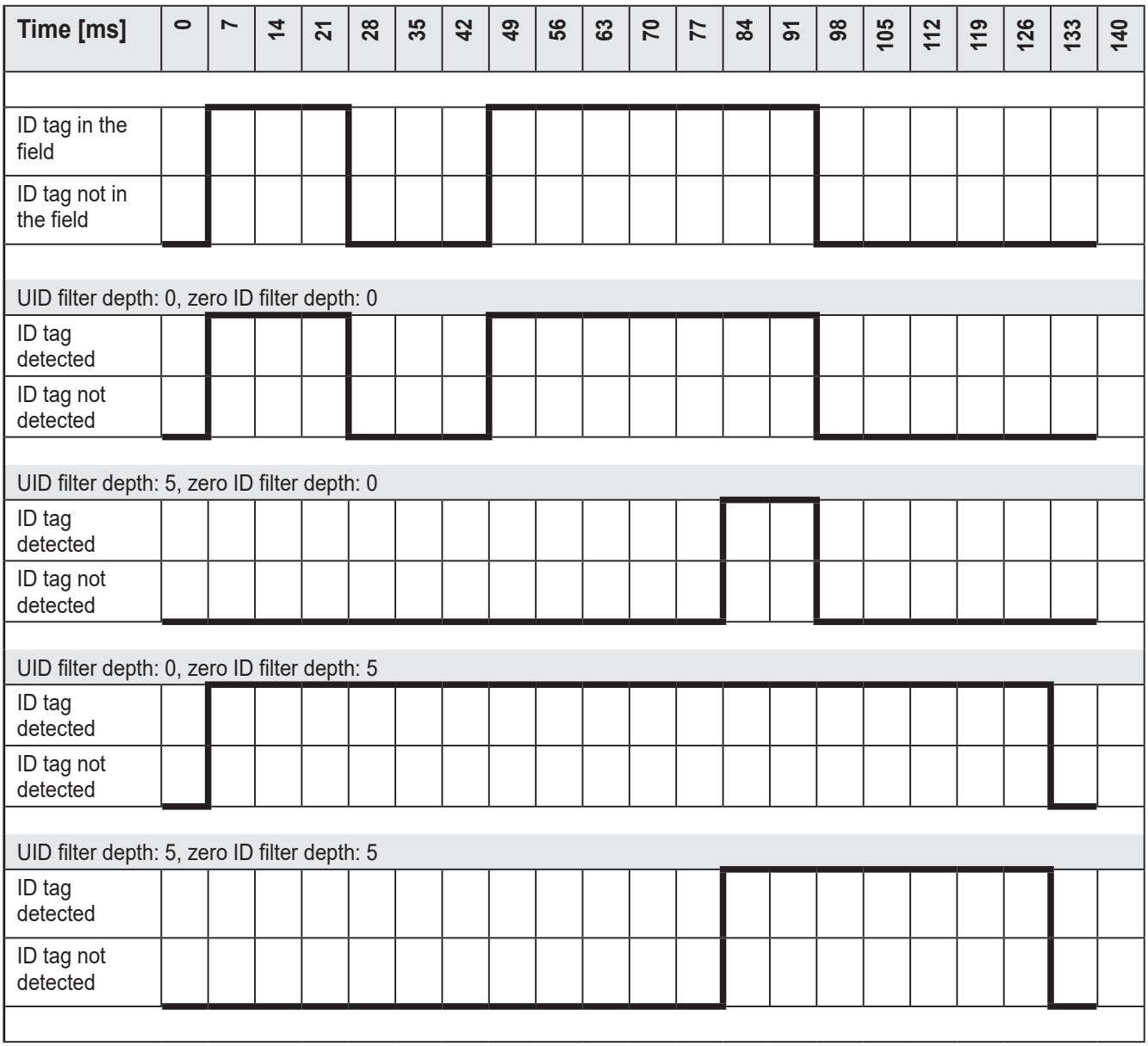
The following situations result in undesired multiple ID tag detection and reading:

- The ID tag is in the limit range.
- The installation conditions have a negative effect on the electromagnetic field of the device.

Thus, the ID tag is not clearly detected which leads to error messages while reading or writing via PDOs. The objects "UID filter depth" and "Zero ID filter depth" allow for error message filtering.

The following values have proven their worth in practice:

- "0" to "5" for dynamic applications (rapidly passing ID tags)
- ">5" for static applications



### 9.16.1 Object UID filter depth

The "UID filter depth" object (0x4603/0x00) allows for determining the number of successful ID tag detections to be executed by the device. The ID tag will only be considered on the CAN bus as having been detected (ID tag present) once the set number has been reached.

The value "0" deactivates the filter. The values ">0" delay the "ID tag present" bit by 7 ms. Thus a switch-on delay of the ID tag value is generated. The detection in the limit range stabilises as no value will be provided as long as the ID tag has not been steadily detected.



### 9.16.2 Object zero ID filter depth

The "Zero ID filter depth" object (0x4605/0x00) allows for determining the number of unsuccessful ID tag detections to be executed by the device. The ID tag will no longer be considered on the CAN bus as present (ID tag present) once the set number has been reached.

The value "0" deactivates the filter. The values ">0" delay the reset of the "ID tag present" bit by 7 ms. Thus a switch-off delay of the ID tag value is generated. The detection in the limit range stabilises as no value will be provided as long as the ID tag does not remain steadily undetected.

## 10 Data transfer with the ID tag

### 10.1 Read UID (Unique Identification Number) of the ID tag

The UID of the tag is available in object 0x2180 as soon as an ID tag is within the reading range of the device. If no ID tag is available, the value 0x0000000000000000 is entered.

If the object is mapped on a TPDO, transmission is event-controlled as soon as an ID tag enters the reading range or is removed from the reading field.

For 32-bit controllers, the objects 0x2190 and 0x2191 must be used instead of object 0x2180.

### 10.2 Read data from the ID tag via PDO transfer

The transfer of the PDO data from the ID tag may be event-controlled. That means that the configured TPDOs are automatically transmitted by the device when the data change. This is the case, for example, when a new ID tag is detected in the detection range of the device. The data is automatically read by the ID tag and transferred by means of the TPDOs via the CAN bus.

The data that was read by the ID tag and assigned to a TPDO is in the object 0x220A with the subindexes 0x01-0x40.

Only that data is read by the ID tag that is assigned to a TPDO. Data objects that are not assigned are not updated automatically.

There are two objects for each data object that are used for configuration: 0x2200 (address read start point) and 0x2201 (read length) with subindexes matching the data object. The start address in the user data area of the ID tag and length of the files to be read are set in the objects.

For 32-bit controllers, the object 0x220B must be used instead of object 0x220A. The maximum data length is restricted to 32-bit data (4 bytes).

If the configured data length is smaller than the data length of the object used (64 bits or 32 bits), the remaining bits are filled with 0.

Max. 64 bits or 32 bits can be transmitted in one TPDO. If larger data volumes are to be transferred, more TPDOs have to be assigned and the respective data objects are to be configured.

### 10.2.1 Example 1

The data range 0x10 to 0x18 (8 bytes) is to be transferred with the 2nd TPDO.

| | Settings for PDO mapping | Object directory | | |
|---|---|---|---|---|
| **TPDO** | **COB** | **Object index** | **Object subindex** | **Object length** |
| 2 | Node ID + 0x0280 | 0x220A | 0x01 | 0x40 |

| Object directory | | | |
|---|---|---|---|
| **Index** | **Subindex** | **Name (object)** | **Value** |
| 0x2200 | 0x01 | Address read start point<br>(start of the address range on the ID tag to be read) | 0x10 |
| 0x2201 | 0x01 | Read length<br>(length of the memory range on the ID tag to be read; max. 8 bytes) | 0x08 |

### 10.2.2 Example 2

The data range 0x44 to 0x48 (4 bytes) is to be transferred with the 6th TPDO.

| | Settings for PDO mapping | Object directory | | |
|---|---|---|---|---|
| **TPDO** | **COB** | **Object index** | **Object subindex** | **Object length** |
| 6 | Node ID + 0x0680 | 0x220A | 0x05 | 0x40 |

| Object directory | | | |
|---|---|---|---|
| **Index** | **Subindex** | **Name (object)** | **Value** |
| 0x2200 | 0x05 | Address read start point<br>(start of the address range on the ID tag to be read) | 0x44 |

| Object directory | | | |
|---|---|---|---|
| **Index** | **Subindex** | **Name (object)** | **Value** |
| 0x2201 | 0x05 | Read length<br>(length of the memory range on the ID tag to be read; max. 8 bytes) | 0x04 |

## 10.3 Write data to the ID tag via PDO transfer

To write data to an ID tag via PDO transfer an RPDO must be assigned to the object 0x230A with a subindex in the range from 0x01 to 0x40. The address of the ID tag user data range to which the data is to be written is defined in object 0x2300. The subindexes of these objects have to be identical.



The ID tag is written to after the data was written to the RPDO and the respective bit was changed in the "Write trigger" object (OD index 0x230F, subindex 0x00).

| | **MSB** | | | | | | | | **LSB** |
|---|---|---|---|---|---|---|---|---|---|
| **Bit** | 63 | 62 | 61 | .. | .. | .. | 2 | 1 | 0 |
| **Trigger** | tr64 | tr63 | tr62 | .. | .. | .. | tr3 | tr2 | tr1 |
| **Default value** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| **Trigger** | **Description** |
|---|---|
| tr64 | trigger for ID tag data 64 (0x230A/0x40) |
| tr63 | trigger for ID tag data 63 (0x230A/0x3F) |
| tr62 | trigger for ID tag data 62 (0x230A/0x3E) |
| tr61 | trigger for ID tag data 61 (0x230A/0x3D) |
| tr60 | trigger for ID tag data 60 (0x230A/0x3C) |
| tr59 | trigger for ID tag data 59 (0x230A/0x3B) |
| tr58 | trigger for ID tag data 58 (0x230A/0x3A) |
| … | … |
| tr6 | trigger for ID tag data 6 (0x230A/0x6) |
| tr5 | trigger for ID tag data 5 (0x230A/0x5) |
| tr4 | trigger for ID tag data 4 (0x230A/0x4) |
| tr3 | trigger for ID tag data 3 (0x230A/0x3) |
| tr2 | trigger for ID tag data 2 (0x230A/0x2) |
| tr1 | trigger for ID tag data 1 (0x230A/0x1) |

The writing process is always made with the bit change of the respective bit (0->1 or 1->0). Ideally, the object "Write trigger" (OD index 0x230F, subindex 0x00) is assigned to an RDPO. On delivery, the object "Write trigger" is assigned to the first RPDO.

Automatic writing of data can be activated with the object "Auto write" (OD index 0x2302). As soon as a tag is in the detection range, the last data is written to the ID tag.

Only data up to the configured data length is written to the ID tag. Subsequent data is ignored. If more than 8 bytes (4 bytes for 32-bit data objects) are transferred, more RPDOs have to be assigned and the respective data objects have to be configured.

For 32-bit controllers, the object 0x230B must be used instead of object 0x230A. The maximum data length is restricted to 32-bit data (4 bytes).

The trigger is divided between the objects 0x231E and 0x231F. The object 0x231E contains the triggers for ID data 33 to 64. The object 0x231F contains the triggers for ID data 1 to 32.

### 10.3.1 Example 1

The data range 0x10 to 0x18 (8 bytes) is to be transferred with the 2nd RPDO.

| | **Settings for PDO mapping** | **Object directory** | | |
|---|---|---|---|---|
| **RPDO** | **COB** | **Object Index** | **Object subindex** | **Object length** |
| 2 | Node ID + 0x0200 | 0x230A | 0x01 | 0x40 |

| **Object directory** | | | |
|---|---|---|---|
| **Index** | **Subindex** | **Name (object)** | **Value** |
| 0x2300 | 0x01 | Address read start point<br>(start of the address range on the ID tag to be read) | 0x10 |
| 0x2301 | 0x01 | Read length<br>(length of the memory range on the ID tag to be read; max. 8 bytes) | 0x08 |
| 0x2302 | 0x01 | Auto-write | 0x00 |

Transfer data via RPDO:

| **PDO transfer** | **PDO** | **Data** |
|---|---|---|
| to the device | RPDO 2 | 0x12345678 |

Start write access:

| **PDO transfer** | **PDO** | **Data** |
|---|---|---|
| to the device | RPDO 1 | select bit 0 status |

### 10.3.2 Example 2

The data range 0x44 to 0x48 (4 bytes) is to be transferred with the 6th RPDO. In addition this data is to be written to an ID tag each time it reaches the detection range of the device.

| | PDO mapping settings | Object directory | | |
|---|---|---|---|---|
| **RPDO** | **COB** | **Object Index** | **Object subindex** | **Object length** |
| 6 | Node ID + 0x0600 | 0x230A | 0x05 | 0x40 |

| Object directory | | | |
|---|---|---|---|
| **Index** | **Subindex** | **Name (object)** | **Value** |
| 0x2300 | 0x05 | Address read start point<br>(start of the address range on the ID tag to be read) | 0x44 |
| 0x2301 | 0x05 | Read length<br>(length of the memory range on the ID tag to be read; max. 8 bytes) | 0x04 |
| 0x2302 | 0x05 | Auto-write | 0x01 |

Transfer data via RPDO:

| PDO transfer | PDO | Data |
|---|---|---|
| to the device | RPDO 6 | 0x12340000 |

The data is written to the ID tag when it has reached the detection range.

64-bit data (8 bytes) / 32-bit data (4 bytes) always have to be sent to an RPDO. If the configured data length is smaller than 64 bits / 32 bits, the remaining bits are ignored.

## 10.4 Error handling for PDO transfer

If a read or write access to an ID tag is not possible, the device creates an emergency message on the CAN bus.

The error code can be read from the error register (OD index 0x1001, subindex 0x00) and the predefined error field (OD index 0x1003, subindex 0x01-0x02) (→ 9.7 Error messages).

## 10.5 Read data from the ID tag via SDO transfer

To read data from an ID tag via SDO transfer it is necessary to define the data address and length on the tag. The address must be indicated in object 0x2280 and the data length in object 0x2281.

Then read access can be started from the ID tag via a data transfer to object 0x2282.

### 10.5.1 Example

The data range 0x50 to 0x70 is to be read from the ID tag.

| Object directory | | | |
|---|---|---|---|
| **Index** | **Subindex** | **Name (object)** | **Value** |
| 0x2280 | 0x00 | Address read start point<br>(start of the address range on the ID tag to be read) | 0x50 |
| 0x2281 | 0x00 | Read length<br>(length of the memory range on the ID tag to be read; max. 8 bytes) | 0x20 |

Transfer is started via reading the object 0x2282, subindex 0x00.

The data is transferred in one piece as domain data type. Up to a data length of 4 bytes transfer is effected as expedited transfer; longer data lengths as segmented transfer.

The recipient must be prepared for temporary storage and processing of the data.

## 10.6 Write data to the ID tag via SDO transfer

To write data to an ID tag via SDO transfer it is necessary to define the data address and length on the ID tag.

The address must be indicated in object 0x2380 and the data length in object 0x2381. Then the write access to the tag can be started via a data transfer to object 0x2382.

### 10.6.1 Example

The data range 0x34 to 0x38 is to be transferred to the ID tag.

| Object directory | | | |
|---|---|---|---|
| **Index** | **Subindex** | **Name (object)** | **Value** |
| 0x2380 | 0x00 | Address write start point<br>(start of the address range on the ID tag to be written) | 0x34 |
| 0x2381 | 0x00 | Write length<br>(length of the memory range on the ID tag to be written) | 0x03 |
| 0x2382 | 0x00 | ID tag data<br>(data to be written to the ID tag) | 0x01020304 |

The data is transferred in one piece as domain data type. Up to a data length of 4 bytes transfer is effected as expedited transfer; longer data lengths as segmented transfer.

The transmitter must be able to provide the indicated data length.

## 10.7 Lock data ranges on the ID tag via SDO transfer

The data ranges of the ID tag can be write-protected.

The write protection of a data range cannot be removed.

The start address of the data range to be protected is stored in the object "Address lock start point" (OD index 0x2480). In addition, the data range length is stored in the object "Write length" (OD index 0x2481).

The start address must be identical with the start address of a storage block on the ID tag. The length must be a multiple of the length of a storage block on the ID tag.



To activate the write protection "1” is written on the trigger (OD index 0x2482).

### 10.7.1 Example

The data range 0x04 to 0x0C is to be write-protected for an ID tag of block size 4 (2 blocks or 8 bytes).

| Object directory | | | |
|---|---|---|---|
| **Index** | **Subindex** | **Name (object)** | **Value** |
| 0x2480 | 0x00 | Address lock start point<br>(start of the address range on the ID tag to be locked) | 0x04 |
| 0x2481 | 0x00 | Write length<br>(length of the memory range on the ID tag to be locked) | 0x08 |
| 0x2482 | 0x00 | ID tag lock trigger | 0x01 |

## 10.8 Error handling for SDO transfer

SDO transfers are acknowledged transfers. If there is an error during transfer or during actions caused by the transfer, an error is signalled after the SDO transfer.

| SDO error code | Description | Possible cause |
|---|---|---|
| 0x05030000 | Toggle bit unchanged. | |
| 0x05040000 | SDO protocol expired. | |
| 0x05040001 | Client/server command specifier not valid or unknown. | |
| 0x05040002 | Invalid block size (block mode only). | |
| 0x05040003 | Invalid sequence number (block mode only). | |
| 0x05040004 | CRC error (block mode only). | |
| 0x05040005 | Out of memory. | |
| 0x06010000 | Access to the object is not supported. | |
| 0x06010001 | Attempt to read a write only object. | |
| 0x06010002 | Attempt to write a read only object. | |

| SDO error code | Description | Possible cause |
|---|---|---|
| 0x06020000 | Object does not exist in the object dictionary. | |
| 0x06040041 | Object cannot be mapped to the PDO. | |
| 0x06040042 | The number and length of the objects to be mapped would exceed PDO length. | |
| 0x06040043 | Reason: general parameter incompatibility. | |
| 0x06040047 | General parameter incompatibility in the device. | |
| 0x06060000 | Access failed due to a hardware error. | |
| 0x06070010 | Data type does not match, length of the service parameter does not match. | |
| 0x06070012 | Data type does not match; service parameter too long. | |
| 0x06070013 | Data type does not match; service parameter too short. | |
| 0x06090011 | Subindex does not exist. | |
| 0x06090030 | Invalid value for parameter (download only). | |
| 0x06090031 | Value of written parameter is too high (download only). | |
| 0x06090032 | Value of written parameter is too low (download only). | |
| 0x06090036 | Maximum value is lower than minimum value. | |
| 0x060A0023 | Resource not available: SDO connection. | |
| 0x08000000 | General error. | |
| 0x08000020 | Data cannot be transferred to the application or be stored. | Error read or write access of the ID tag. Detailed information in the device status object (0x2150). |
| 0x08000021 | Data cannot be transferred to the application or be stored due to a local controller. | |
| 0x08000022 | Data cannot be transferred to the application or stored due to the current device status. | |
| 0x08000023 | The dynamic generation of the object directory fails or no object directory is present (e.g. object directory is generated from the file and the generation fails because of a file error). | |
| 0x08000024 | No data available. | data length = 0 |

# 11 EDS file

The EDS file serves as a template for different configurations of a device type. The EDS file is turned into a DCF file which contains device configurations, object values, node ID and bit rate.

CANopen configuration tools are available for the configuration of the CANopen network and the devices.

The EDS files are available on ifm's website: www.ifm.com

Contents of the EDS file:

- Communication functions and objects (to CANopen profile DS-301)
- Manufacturer-specific objects

The installation of the EDS file depends on the configuration tool.

► Contact the manufacturer of the controller for more information.

The EDS files are supplied with 64-bit or 32-bit data types. The controller determines whether 64-bit or 32-bit data types can be processed.

► Select the EDS file appropriate to the controller.



## 12 Maintenance, repair and disposal

► Do not open the housing as the device does not contain any components which can be maintained by the user. The device must only be repaired by the manufacturer.

► Dispose of the device in accordance with the national environmental regulations.

# 13 Glossary

| Term | Description |
|---|---|
| 0b … | binary value (for bit coding), e.g. 0b0001 0000 |
| 0x … | hexadecimal value, e.g. 0x64 (= 100 decimal) |
| AFI | indication of the application range of the ID tag |
| CAN | Controller Area Network (bus system for use in mobile vehicles) |
| CAN_H | CAN high; CAN connection / CAN cable with a high voltage level |
| CAN_L | CAN low; CAN connection / CAN cable with a low voltage level |
| CANopen | CAN-based network protocol on the application level with an open configuration interface (object directory) |
| CiA | CAN in Automation e.V. (user and manufacturer organisation in Germany/Erlangen, definition and control body for CAN and CAN-based network protocols) |
| COB | CANopen communication object: PDO, SDO, EMCY, ... |
| COB ID | communication object identification number for assignment of the data packages in the CANopen network |
| DSFID | identification number for the assignment of the data structure on the tag |
| EDS | electronic data sheet |
| EMCY object | emergency object (alarm message; device signals an error) |
| Emergency messages | messages on the CANopen bus to communicate errors |
| Error reg | error register (entry with an error code) |
| Heartbeat | Configurable cyclic monitoring among network participants. In contrast to "node guarding" no superior NMT master is required. |
| ID | Identifier characterising a CAN message. The numerical value of the ID also contains a priority concerning the bus access (ID 0 = highest priority). |
| Identifier | see ID |
| LSS | procedure to set basic device settings |
| NMT | network management |
| NODE-ID | unambiguous number of a participant in the CANopen network |
| Object / OBJ | term for data/messages which can be exchanged in the CANopen network |
| OD | object directory |
| PDO | Process Data Object; in the CANopen network to transfer process data in real time such as motor speed. PDOs have a higher priority than SDOs; in contrast to the SDOs they are transferred without confirmation. PDOs consist of a CAN message with identifier and up to 8 bytes of user data. |
| PDO mapping | describes the application data transferred with a PDO |
| ro | unidirectional; read only |
| RPDO | process data object, received by the device |
| RSSI | signal strength |
| rw | bidirectional; read / write |

| Term | Description |
|---|---|
| SDO | With this object direct access to the object directory of a network participant is possible (read/write). An SDO can consist of several CAN messages. The transfer of the individual messages is confirmed by the addressed participant. With the SDOs, devices can be configured and parameters can be set. |
| SYNC | The SYNC telegram initiates the synchronised transmission of process data. |
| TPDO | process data object, sent by the device |
| UID | unique identification number of an ID tag |
| wo | unidirectional, write only |