CANOPEN KNOW-HOW

Basics of the CANopen and CANopen FD protocol

Mechanisms, Functions, Parameters and Implementation

HMS

CANopen and CANopen FD protocol standard and mechanisms

CANopen is a higher-layer protocol based on CAN (Controller Area Network), which enables the communication between devices of different manufacturers and guarantees interchangeability of devices.

The CANopen profile family defines a standardized application for distributed systems based on CAN. CANopen was developed within the CAN-in-Automation (CiA) international users' and manufacturers' group and is standardized as CENELEC EN 50325-4 since December 2002. Soon after its initial release in 1996, CANopen found broad acceptance, especially in Europe where it can be considered the leading standard for CAN based system solutions.

Functionality, parameters and access to process data from standard devices, such as I/O modules, drives, controllers or encoders, are defined by device profiles, so devices from different manufacturers can be accessed via the bus in exactly the same manner. As a result, CANopen offers a very high degree of vendor independence, as devices are interoperable and exchangeable. On one hand CANopen is standardized, but on the other, it is open to a near unlimited field of applications, including:

- Machine control
- Factory automation
- Laboratory automation
- Transportation and traffic
- Utility vehicles
- Building automation
- Medical Systems

CONTENT

CANOPEN / CANOPEN FD INTRODUCTION

CANopen provides service and protocol specifications and device and application profile specifications. Devices compliant with this set of specifications are interoperable with other CANopen devices; implementing the same functionality even enables the interchangeability of products.

CANopen was developed in 1993 within a European research project under the leadership of Bosch, with a research team comprising universities and companies from different European countries.

In 1994, the developed CANopen specifications were handed over to the CAN in Automation (CiA) international users' and manufacturer's group. Since then, the non-profit association has maintained the documents and has developed and enhanced the communication system. Nowadays, the CANopen specification comprises more than 20,000 pages of specification, including device and application profiles for different industries.

Besides embedded machine control, CANopen was initially used in medical devices, such x-ray apparatus, computer-tomography systems, etc. In the late 90s, CANopen made its way into off-road vehicles, followed by users in the elevator rail vehicle industries. Applications we also seen in deeply embedded control systems, where a typical application might be the control of sliding or rotating doors. Another example of deeply embedded networks sees CANopen used as backbone bus system in modular I/O devices.

Applications for CANopen are nearly limitless. Wherever more than two micro-controller-based units need to communicate, CANopen is a candidate. The robustness and reliability of the CAN lower layers, combined with the flexibility of the CANopen application layer and the interoperability provided by the CANopen profiles, makes CANopen one of the most successful embedded control networks.

One of the reasons for CANopen's success is the relative stability of the base services and protocols. There has been just one major improvement from version 3.0 to version 4.X – not considering the two first versions in the prototyping phase (1994 to 1995). This has reduced the investment overheads for CANopen device suppliers and system designers.

CANopen mapped on Classical CAN hardware does has some limitations: The bandwidth at given network length is limited, as well as the payload in a single CAN frame. These limits can be overcome using the CAN FD data link lower layers, which support higher bit-rates (more than 1 Mbit/s) and the CAN FD frames can contain up to 64 bytes instead of 8 up to byte in Classical CAN frames.

The CANopen FD application layer – released in 2017 – makes use of this larger payload by introducing new communication services. CANopen FD is the successor of Classic CANopen providing a similar robustness and reliability and with the enhanced communication services CANopen FD is ready for new applications, including Industrial Internet of Things (IIoT) support.

CANopen basics

CANOPEN APPLICATION LAYER BASICS

CiA 301 describes the basic communication services and protocols mapped to the Classical CAN data link layer. CANopen services are also mapped to other communication technologies, such as EtherCAT and Powerlink.

The specified communication services and the related protocols (given in brackets) comprise:

- Network management (NMT and Heartbeat)
- Device configuration (SDO)
- Real-time transmission of process data (PDO)
- Node synchronization (SYNC and TIME)
- Diagnostics (EMCY and SDO)

The related CANopen protocols require the assignment of a unique 7-bit node-ID by the system designer, in order to guarantee that all protocols use a unique CAN-ID. This can be achieved by different methods. Most common is the assignment by means of DIP switches. But there are methods, such as dedicated configuration interfaces (e.g. a display). If the CAN interface of the CANopen device is used, the layer setting services (LSS) as defined in CiA 305 should be used.

CANopen also requires the implementation of an object dictionary, which lists all parameters representing the CANopen device functionality, including process data, configuration options, and diagnostic information. This object dictionary is well structured, and each parameter is addressable by means of a 16-bit index and an 8-bit sub-index. This 24-bit address is used by some communication services as a multiplexer to overcome the limitation of just 2048 identifiers provided by the CAN data link layer. This is effectively a prolongation of the 11-bit CAN-ID used by default for all CANopen protocols.

Object dictionary structure

The 16-bit index range is structured as follows:

0000_{16}	Reserved
0001_{16} to $025F_{16}$	Data type parameters
0260_{16} to $0FFF_{16}$	Reserved
1000_{16} to $1FFF_{16}$	Communication parameters
2000_{16} to $5FFF_{16}$	Manufacturer-specific parameters
6000_{16} to $67FF_{16}$	Logic device 1 parameters
6800_{16} to $6FFF_{16}$	Logic device 2 parameters
7000_{16} to $77FF_{16}$	Logic device 3 parameters
7800_{16} to $7FFF_{16}$	Logic device 4 parameters
8000_{16} to $87FF_{16}$	Logic device 5 parameters
8800_{16} to $8FFF_{16}$	Logic device 6 parameters
9000_{16} to $97FF_{16}$	Logic device 7 parameters
9800_{16} to $9FFF_{16}$	Logic device 8 parameters
$A000_{16}$ to $AFFF_{16}$	Network variables
$B000_{16}$ to $BFFF_{16}$	System variables
$C000_{16}$ to $FFFF_{16}$	Reserved

Each of these parameters can have up to 256 sub-parameters addressable by means of the 8-bit sub-index. The sub-parameter 00_{16} is used to indicate the highest implemented sub-parameter in case of arrays (all sub-parameters are of the same data type) or records (sub-parameters are of different data types). Parameters specified as variables support just the sub-parameter 00_{16}.
The CANopen object dictionary supports up to eight logical devices, as a result you can implement multiple device profiles in a single CANopen device. This could be a motion controller with eight motor instances or a motion controller with additional I/O functionality.

List of abbreviations

CAN	Controller area network
EMCY	Emergency object
ID	Identifier
I/O	Input/output
LSS	Layer setting services
NMT	Network management
PDO	Process data object
SDO	Service data object
SYNC	Synchronization object
TIME	Network time object

CANOPEN AND THE OSI REFERENCE MODEL

The Open System Interconnection (OSI) reference model from ISO specifies seven layers. The CANopen application layer and communication profile – as specified in CiA 301 or EN 50325-4 (equivalent to CiA 301 version 4.0) – mainly cover the transport layer and the application layer. The presentation, session, and network layers are not used.

The data link layer complies with ISO 11898-1 and uses data frames in CBFF (Classical Base Frame Format) by default, and optionally data frames in CEFF (Classical Extended Frame Format). Remote frames are allowed but are not recommended at all.

By default, CANopen uses the CAN physical layer as defined in ISO 11898-1 (physical signaling sub-layer) and in ISO 11898-2 (physical media access sub-layer). This enables bit-rates up to 1 Mbit/s. In order to improve interoperability, CiA 301 limits the bit-rates to the following values and sample-point ranges (given in percentage of the bit-time):

- 1 Mbit/s (75 % to 90 %)
- 800 kbit/s (75 % to 90 %)
- 500 kbit/s (85 % to 90 %)
- 250 kbit/s (85 % to 90 %)
- 125 kbit/s (85 % to 90 %)
- 50 kbit/s (85 % to 90 %)

It is recommended that daisy-chained line topologies or line topologies with short stubs are used. Bus ends of the network cable needs to be terminated by resistors (nominally 120 Ω each). The network length at a given bit-rate depends not only on the configured sample, but also, for example, on the cables used and non-terminated stubs. In 1-Mbit/s networks you can achieve about 25 m. Using 500 kbit/s, network lengths of up to 125 m are possible. In 250 m networks, bit-rates of up to 250 kbit/s can be reached. Setting the bit-rate to 125 kbit/s allows up to 500 m and at 50 kbit/s, the maximum length is 1 km.

Optionally, transceivers compliant to ISO 11898-3 with low-power capabilities and fault-tolerant functions are allowed, but not recommended for new designs. They are limited to 125 kbit/s.

OSI reference model

Application level		User program(s)
Data level		CiA 4XX: Device and application profiles
OSI layers		
	Application layer	CiA 301: NMT, Heartbeat, SDO, PDO, SYNC, EMCY, TIME
	Presentation layer	CiA 301: Data types ans encoding rules
	Session layer	Not applicable
	Transport layer	CiA 301: Segmented SDO
	Network layer	(CiA 302-7: SDO and EMCY routing, PDO bridging)*
	Data link layer	ISO 11898-1
	Physical layer	ISO 11898-2, CiA 301 (bit-timing), CiA 303-1 (cable and connectors)

* Only necessary in multiple CANopen network architectures

The network designer must assign unique CANopen node-IDs to each connected CANopen device. Additionally, all CANopen nodes must use the same bit-rate.

The CANopen profile specifications defining the process data, configuration parameters, and diagnostic information are above the OSI reference model. This also includes the mapping of process data into PDOs.

CiA 302-7 specifies a network layer. The described protocols allow the access of CANopen devices from another CANopen network segment (remote SDO services). This can be used to configure a complex CANopen system, comprising several segments, from a single point. This network layer can be also used for diagnosis purposes (remote EMCY services).

Repeater, bridge/switch, router, and gateways

CANopen applications can make use of CAN repeaters (OSI layer 1). Repeaters enable more CAN nodes in one CANopen segment or longer network lengths due to the refreshing of the bus signals.

CAN bridges or switches (OSI layer 2) can be used to separate CANopen network systems into different segments, in order to limit impacts or to reduce busloads. For this purpose, the system variables as specified in CiA 302-7 are used.

Routers (OSI layer 3), compliant to CiA 302-7, can be used to forward SDO messages and EMCY messages to other network segments.

If connections to other network technologies are needed, gateways can be used. The CiA 309 series specifies the access from TCP/IP-based networks to CANopen networks. This includes PROFINET (CiA 309-4) and Modbus-TCP (CiA 309-2). CANopen also provides gateway specifications for AS-i (CiA 446) and IO-Link (CiA 463 series, under development).

This means, homogeneous CANopen network systems and heterogeneous network systems are supported by CANopen specifications. The CANopen router function requires the assignment of a unique 8-bit network ID, which, considering that 127 nodes can reside in one network segment, allows the addressing of thousands of CANopen devices in network system.

CANOPEN DEVICE MODEL

The CiA 301 specification uses a device model as shown below. It includes a CAN entity, which comprises the CAN transceiver and the CAN protocol controller. The CAN protocol controller is in most cases on-chip of the micro-controller – sometimes named host controller. The CANopen protocol stack implements the CANopen protocols and the CANopen object dictionary. And there is the profile and application program of the CANopen device, which may comply with one of the CANopen profiles specified by CiA. Of course, the CANopen device may implement just manufacturer-specific profiles.

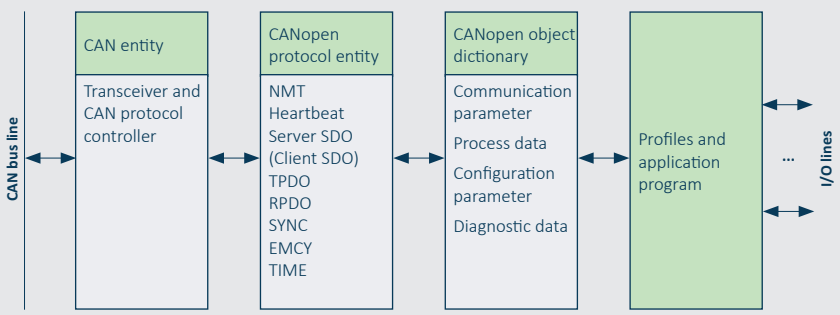
The CANopen device's communication and application parameters accessible by means of the object dictionary are represented electronically in the Electronic Data Sheet (EDS). There are two versions: one is using an ASCII format (CiA 306) and another is using an XML schema (CiA 311). Both files provide the parameter and sub-parameter attributes. This includes index and sub-index, name, object code, data type, category (or entry category), access, PDO mapping, value range, and default value. EDS are used to teach device and system configuration tools the provided functionality. Such tools can also be implemented in the host controller with NMT master capability.

NETWORK MANAGEMENT (NMT)

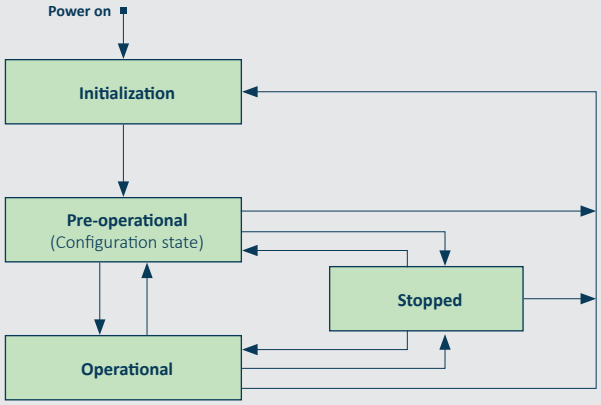
CANopen network management is based on a master/slave approach. The device with NMT master functionality controls the CANopen NMT slave devices. Each NMT slave can be in one of three static states:

- Pre-operational: All CANopen services can be used, except PDO services,
- Operational: All CANopen services can be used, or
- Stopped: No CANopen services can be used, except NMT and Heartbeat

There are additional temporary states, which the device transmits automatically after power-on and after reset. The state diagram provides details on



The CANopen device model



NMT slave state machine

the possible state transitions. Those transitions are commended by the NMT master or device internally by the application.

The NMT commands from the NMT master devices are confirmed at the application program level by means of a Heartbeat message. The NMT command message is a two-byte message. One byte contains the command and the other the addressed node-ID. A node-ID of “0” indicates a broadcast command, meaning that all nodes shall perform the command. It is mapped to the CAN data frame with CAN-ID “0”, which is the highest prior one. The above-mentioned confirmation is a one-byte message providing the current state (pre-operational, operational, or stopped) of the related NMT slave device. It is mapped, depending on the node-ID of the transmitting device, to a CAN data frame with a CAN-ID of 800_{16} plus node-ID. The Heartbeat message is sent periodically with the user-configurable heartbeat-producer-time given in the object dictionary.

After power-on, the CANopen NMT slave transits automatically into the NMT pre-operational state and waits to be configured (optional) and started by the NMT master device. It is also possible to configure an NMT slave device to be self-starting. This is required by some CANopen applications. There are two reset commands: one simply resets the communication parameters to the default or the configured values; the other resets the communication and the profile parameters. After the reset procedure, the NMT slave device transitions automatically to the NMT pre-operation state.

The NMT stopped state can be used to stop a device communicating SDO and PDO messages. It just accepts the NMT message and produces the Heartbeat message. In some cases, it is desirable to have the devices not transmit any messages. To achieve this, the heartbeat-producer-time in the object dictionary needs to be configured to “0”.

The legacy option to remotely request the Heartbeat message by means of a CAN remote frame (so-called Node/Life guarding) is no longer recommended. This approach also used a life-timer implemented in each NMT slave device. When it expires without receiving the appropriate CAN remote frame from the NMT master device, the NMT slave considers that the NMT master device is no longer available, and behaves as programmed for this case.

NMT flying master

In mission-critical applications, a single entity is not acceptable. As a result, CiA 302-3 specifies the NMT flying master approach.

There are several services that allow a second NMT master capable device to function as an NMT master device, when the original NMT master is not available. When the original NMT master comes back, it requests to become the active NMT master. This NMT flying master functionality is used in maritime electronics and medical devices. Sub-sea trees on the ocean floor implement redundant NMT master devices too, using the NMT flying master protocols.

HEARTBEAT SERVICES

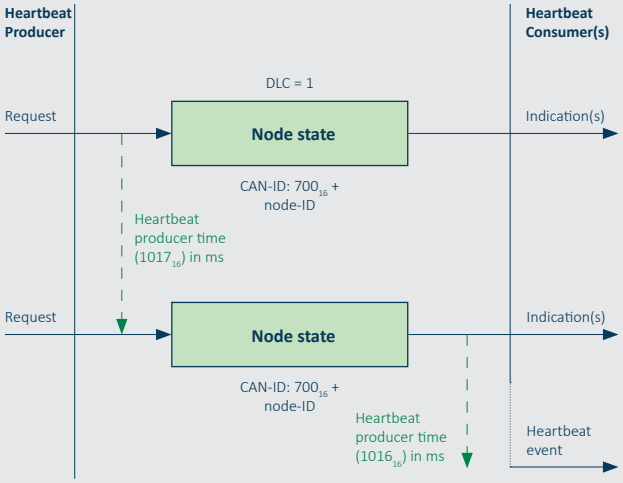
CANopen network management uses a Heartbeat message as a confirmation of the NMT command sent by the NMT master device. Additionally, any CANopen device can use it to check the availability of any other CANopen device. This is necessary in all cases in which CANopen devices transmit PDO messages only on state-of-change events. The subscriber of such PDO messages would not know if there is no event or if the device is no longer available. The reception of a Heartbeat message from the relevant device indicates that it is still alive.

The transmission period is configurable by means of the heartbeat-producer-time parameter in the object dictionary of the transmitting device. The indication that a device is missing can be configured in any interested device by means of the heartbeat-consumer-time array. As a rule of thumb, the consumer time should be twice the producer time. The producer and consumer time configurations are highly application specific.

In some applications, devices crosscheck each other by consuming the Heartbeat messages. Sensors normally do not consume Heartbeat messages they just produce their PDO messages and their Heartbeat messages to indicate, respectively, that they are still alive and to confirm the NMT command.

System designers are responsible for the configuration of the Heartbeat timing. There could be application-specific requirements that a device missing an essential device stops the production of its own Heartbeat message. System designers should take care that the overall availability is still sufficient.

The 1-byte Heartbeat message uses a CAN data frame with the ID 700_{16} plus producer node-ID. The one-byte content indicates the NMT status. This means, the consumer can also detect that a device has not sent PDO messages because of its NMT state. This can be evaluated in the consumer's application program.



Node state values:
04_{16} (Stopped), 05_{16} (Operational), $7F_{16}$ (Pre-operational)

Heartbeat protocol: When the configured heartbeat-consumer-time elapses, the application receives an indication and may stop the production of its own Heartbeat message, because a missing CANopen device could be critical for functionality.

Heartbeat message content

The Heartbeat message is mapped to a CAN data frame with a 1-byte data field. This byte provides the NMT status of the CANopen device. It could be in pre-operational state ($7F_{16}$), operational state (05_{16}), or stopped state (04_{16}). In the pre-operational state, the CANopen device does not process PDOs and in the stopped state it just supports the NMT message and sends its Heartbeat. In the operational state, all CANopen functions are provided.

If the CAN data frame contains 00_{16}, it is interpreted as Boot-up message indicating that this CANopen device has just entered the network. This happens after initial power-on, a power-cycle, and after application and communication reset. The next CAN data frame with the same CAN-ID and content not equal to 00_{16} is interpreted as Heartbeat message. The NMT master device uses the Heartbeat message as a confirmation of its NMT commands.

SDO SERVICES

CiA 301 specifies several SDO services. The main purpose of these services is to read and write data to one dedicated CANopen object dictionary address.

The SDO client always has the initiative, sending an SDO read or write request to the desired SDO server. Therefore, SDO services are always confirmed. Read requests are confirmed by the SDO data received from the SDO server. Write requests are confirmed by means of SDO server messages indicating that the SDO data is stored in the CANopen object dictionary.

In case the SDO service cannot be handled correctly, either the SDO server or the SDO client may send an SDO-abort message. The SDO-abort message provides a 4-byte abort code, giving the user information about the reason. It is not very helpful to provide an abort code reporting a general error. Therefore, CiA 301 specifies for some cases, which abort code needs to be used.

SDO messages have a 4-byte SDO protocol overhead containing the kind of service and the 24-bit address of the object dictionary parameter to be written or to be read. There are also some bits in specific SDO protocol control bits, for example, a toggle-bit indicating that the CAN data link layer has not duplicated the transmitted message.

Some SDO services allow reading or writing of data of any length. Because the CAN data frames provide just an 8-byte payload (data field), the related SDO transport layer protocol segments the SDO data on the sender side and reassembles them on the receiver side. Each CAN data frame sent by the SDO client is confirmed by the SDO server by means of a CAN data frame.

There are two options to indicate the length of the data to be transferred: the size or an end indication (last segment) both of which are provided in the SDO protocol overhead. In order to reduce the busload, SDO block services can be specified, which do not confirm each individual segment, but instead a block of segments. The block size and number of segments is configurable.

All SDO services are mapped to CAN data frames with an 8-byte length. If the SDO data is less than 4 bytes, no segmentation is needed. If the length exceeds four bytes, one of the appropriate SDO transport protocols is used. In the initial CAN frame, the SDO protocol overhead is four bytes. In the following CAN frames just one byte is used for protocol data. The remaining seven bytes contain SDO data. If the SDO data is not entirely used, this is indicated in the SDO protocol.

For each peer-to-peer SDO channel, client/server relation, two CAN-IDs are needed: one for the client-to-server request and the other for the server-to-client confirmation. Theoretically, all CANopen devices can communicate with all other network nodes by means of SDO services. However, this would require many CAN-IDs. A fully meshed bidirectional SDO communication for all 127 possible nodes would require more than 500 CAN-IDs.

Originally, SDO services were intended to configure or to diagnose a CANopen device. But SDO services can also be used to transmit process data. The system designer should consider that SDO services increase the busload.

SDO protocol examples

There are several SDO protocols specified in CiA 301. In the 1-byte command specifier (CS) of the SDO client, SDO server, and the SDO abort message, three bits are used to indicate the used protocol. In case of a SDO read service to the heartbeat producer time (index 1017_{16} and sub-index 00_{16}) the SDO client sends the following 8-byte value in the CAN data field:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CS: 43_{16}	Index: 1710_{16}		Sub-Index: 00_{16}	(Parameter) data: $00\ 00\ 00\ 00_{16}$			

LSB MSB

The SDO server responds with the following 8-byte value in its SDO server message indicating that the heartbeat producer time is 4000 ms:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CS: $??_{16}$	Index: 1710_{16}		Sub-Index: 00_{16}	(Parameter) data: $A0\ 0F\ 00\ 00_{16}$			

LSB MSB

In case of a SDO write service to the same parameter configuring the heartbeat producer time to 2000 ms, the SDO client sends the following 8-byte value in the CAN data field:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CS: $2B_{16}$	Index: 1710_{16}		Sub-Index: 00_{16}	(Parameter) data: $D0\ 0F\ 00\ 00_{16}$			

LSB MSB

The SDO server confirms the SDO write request with the following 8-byte value in its SDO server message:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
CS: 60_{16}	Index: 1710_{16}		Sub-Index: 00_{16}	(Parameter) data: $D0\ 0F\ 00\ 00_{16}$			

LSB MSB

The SDO abort message always uses the CS value of 80_{16} independent of the source (SDO client or SDO server). This is followed by the index and sub-index as well as the 4-byte abort code.

PDO SERVICES

Process Data Object (PDO) services are used to transmit process data. Process data could, for example, measure values or commands, most of which are time critical. The PDO service is not confirmed, it is a producer/consumer service or, in other words, a publisher/subscriber service.

PDO messages contain up to eight bytes of process data. They are mapped to a single CAN data frame using a CAN-ID determined by the COB-ID parameter given in the CANopen object dictionary. This parameter is part of the PDO communication parameter set, which also comprises the transmission type, which indicates the kind of transmission mode respectively reception mode.

In general, CANopen allows the transmission of PDOs asynchronously and synchronously. The triggering of asynchronous PDOs depends on device-internal events. This can be the change of process data transmitted in this PDO or the elapsing of the event-timer – also part of the PDO communication parameter set. The event-timer enables a periodical transmission of the PDO. But bear in mind, the event-timer is a local timer and not synchronized with the other CANopen device in the network.

Synchronous transmission of PDOs is a unique feature of the CANopen application layer. It is based on the periodical transmission of the SYNC message, which uses a single CAN data frame with the default high-priority CAN-ID of 80_{16}. The SYNC message triggers the transmission of all synchronous PDOs in the CANopen network. This means, all sensors measure their values at the very same moment. They are synchronized and, depending on the internal delay and the priority of the PDO message, these values are transmitted one after the other, but all have been measured simultaneously.

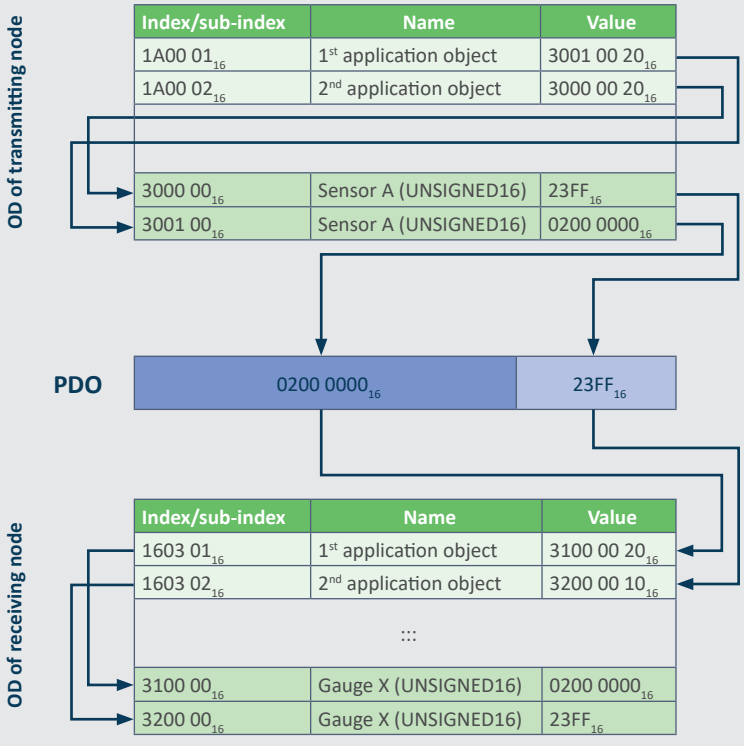
In order to synchronize "fast" and "slow" CANopen devices, synchronous TPDOs can be configured to serve each second up to each 240th SYNC message. But with this mechanism you cannot group synchronous PDOs. For this feature, you need to use the SYNC message with a 1-byte counter. Additionally, the SYNC start value (sub-index 06_{16}) needs to be configured.

Now, you can start one group of synchronous PDOs, for example, at the SYNC counter value of 1 serving each second SYNC message, and another group at the SYNC counter value of 2 also serving each second SYNC message. Of course, you can configure more complex behavior, in order to achieve a stable average bus-load.

In order to synchronize outputs in different devices, the related RPDOs need to be configured as synchronous. This means, the received commands are not performed immediately, but, instead, when the next SYNC message is received. At this time all devices perform the requested actuation in synchronization.

Another unique feature is the PDO inhibit time. This configurable time circumvents the transmission of the PDO for the configured period. This means, a high-priority PDO cannot occupy the entire bandwidth, and lower-priority PDOs can win bus-arbitration. With this feature you can design a predictable transmission of CoS-triggered PDOs.

There is a second PDO parameter set specifying the content of the PDO message. It is possible to map multiple process data into one PDO message. Bit-wise mapping is supported, but not recommended. The related PDO mapping parameter array in the object dictionary provides the object dictionary addresses (16-bit index plus 8-bit sub-index), where the process data value is stored (in case of a Transmit-PDO) or should be stored (in case of a Receive-PDO).



The PDO mapping parameters on the transmitting and the receiving side can differ, the example shows a sensor device transmitting two process data and a display device with virtual gauges receiving the PDO.

The system designer has several opportunities to optimize the PDO communication to the application requirements:

- PDO prioritization: Configuring the COB-ID parameter with an appropriate CAN-ID value changes the priority when accessing the CAN network.
- PDO linking: Configuring the COB-ID parameter of the RPDO to a dedicated CAN-ID value is a subscription of the corresponding TPDO. It is possible that multiple or all CANopen devices subscribe to the same TPDO (multi-cast and broadcast communication)
- PDO scheduling: Configuring the transmission type sub-parameter of a TPDO determines the sending behavior (e.g. change-of-state (CoS), periodically, or synchronously). Configuring the transmission type parameter of a RPDO determines the reception behavior (performing the required action immediately or with the reception of the next SYNC message).
- PDO mapping: Configuring the PDO mapping parameter set determines which process data are mapped into the TPDO or where to enter the received process data (RPDO).

Transmission type values

The transmission type sub-parameter determines the scheduling of TPDO. In the case of a synchronous transmission, the TPDO can be configured to serve each received SYNC message, each second, etc. Remotely requested TPDO should not be used, due to a completely specified behavior (the reaction on CAN remote frames can be implemented differently, for example).

Value	Description
00_{16}	Acyclic synchronous: Triggered when the SYNC message is received and one of the mapped process data has changed its value after the last transmission.
01_{16} to $F0_{16}$	Cyclic synchronous: Triggered when the SYNC message is received (01_{16}), each second SYNC message is received (02_{16}), etc. This supports fast and slow changing process data.
$F1_{16}$ to FB_{16}	Reserved
FC_{16}	Synchronous RTR only: not recommended anymore
FD_{16}	Asynchronous RTR only: not recommended anymore
FE_{16}	Asynchronous: Triggered by an internal event (e.g. change-of-state of one of the mapped process data or elapsing of the event-timer or any other event). The device manufacturer specifies the internal event triggering the TPDO transmission.
FF_{16}	Asynchronous: As before, but the CiA profile specifies the internal event triggering the TDPO transmission.

Transmission type values

PDO communication parameter set

This parameter set is a record. The first sub-parameter (highest sub-index supported) indicates the sub-parameter with the highest number. The COB-ID sub-parameter contains the CAN-ID to be used and some additional protocol control bits. The transmission type sub-parameter determines the triggering (TPDO) or reception (RPDO) behavior). The inhibit-time sub-parameter provides the time in milliseconds when this TPDO is allowed to send it again. The event timer sub-parameter specifies the period in milliseconds of the TPDO transmission respectively the time-out (missing) of a RPDO. The SYNC start value sub-parameter indicates when the synchronous TPDO is transmitted first. The value of this sub-parameter needs to match with the value given in the 1-byte SYNC message.

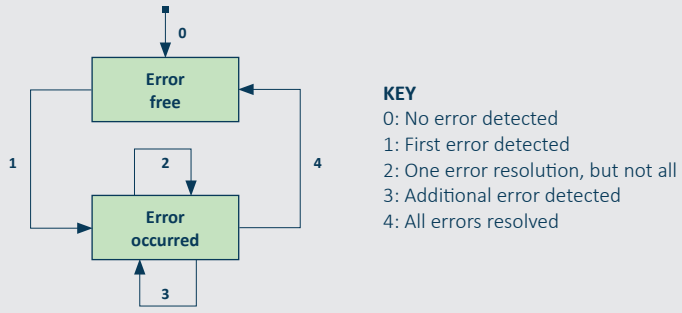
Sub-index	Name	Data type
00_{16}	Highest sub-index sup.	Unsigned8 (BYTE)
01_{16}	COB-ID	Unsigned32 (DWORD)
02_{16}	Transmission type	Unsigned8 (BYTE)
03_{16}	Inhibit-time	Unsigned16 (WORD)
04_{16}	Reserved	Not applicable
05_{16}	Event timer	Unsigned16 (WORD)
06_{16}	SYNC start value	Unsigned8 (BYTE)

PDO communication parameter set

EMCY SERVICE

The EMCY service is mapped to a producer/consumer protocol. The producer sends the EMCY message indicating some detected “errors”. The 8-byte message contains the one-byte error register, the two-byte emergency error code, and five bytes, which are CANopen profile-specific. Some profiles allow using the five bytes for manufacturer-specific purposes.

The error register is a variable listed in the CANopen object dictionary. It controls the Emergency state machine. If any “error” is detected, the CANopen device triggers the EMCY service and sends the EMCY message. If further “errors” occur, the CANopen device remains in the error state. Only if all “errors” are recovered will the CANopen device transition into the error-free state. The error state behavior could be defined by CANopen profiles. In the CiA 401 profile for modular I/O devices, the outputs are set to pre-defined values when the device is in error state.



Error state machine

The detected misbehavior could be regarding the CAN data link (e.g. recovered from bus-off state or error passive mode indication), the CANopen application layer (e.g. PDO length exceeded or RPDO timeout), or the device’s application (e.g. mains voltage or over-temperature). Besides some general emergency error codes, the CANopen profiles specify additional ones.

16-bit error code	8-bit error register ($1001\ 01_{16}$)	Device-profile of manufacturer-specific error information

Structure of the EMCY message: The 16-bit error code is specified in CiA 301 and optionally in the implemented CiA device profile

The system designer needs to configure the EMCY message consumers depending on the application requirements. This is done by means of the emergency consumer parameter listed in the CANopen object dictionary. This array parameter contains all COB-IDs of EMCY messages, which are consumed. NOTE: The COB-ID is a 32-bit value providing the CAN-ID of the data frames to be consumed, as well as three control bits.

The reaction to the received EMCY messages is highly application-specific. By default, the EMCY message is transmitted in a CAN data frame with a high-prior CAN ID: 80_{16} plus node-ID. The system designer assigns a unique node-ID to the CANopen devices. The EMCY COB-ID parameter, which contains the CAN-ID to be used, can be configured by means of an SDO write service to the CANopen object dictionary.

Communication errors

CiA 301 specifies some detailed communication error codes. They are related to the CAN data link layer protocol as well as the CANopen application layer protocols. The implementation of the following error codes is recommended:

- 8110_{16}: CAN overrun (data frame lost)
- 8120_{16}: CAN protocol controller in error passive mode (node cannot indicate bus errors, when it is a receiver)

- 8140_{16}: CAN node recovered from bus-off (may have missed some CAN data frames)
- 8150_{16}: CAN-ID collision (node has detected data frames with CAN-IDs that are assigned to it)

It is recommended that the following CANopen protocol error codes are also supported:

- 8130_{16}: Heartbeat error (one of the supervised CANopen device has not sent its Heartbeat)
- 8210_{16}: PDO not processed due to length error (PDO receiving CANopen device detected a mismatch of the PDO length)
- 8220_{16}: PDO length exceeded (the mapped process data is longer than 8 byte)
- 8240_{16}: Unexpected SYNC message length (SYNC consumer expect another SYNC format with or without SYNC counter)
- 8250_{16}: RPDO timeout (RPDO event timer elapsed)

Value	Description
$00xx_{16}$	No error or reset
$10xx_{16}$	Generic error
$20xx_{16}$	Current
$21xx_{16}$	Current, CANopen device input side
$22xx_{16}$	Current inside the CANopen device
$23xx_{16}$	Current, CANopen device output side
$30xx_{16}$	Voltage
$31xx_{16}$	Mains
$32xx_{16}$	Voltage inside the CANopen device
$33xx_{16}$	Output voltage
$40xx_{16}$	Temperature
$41xx_{16}$	Ambient temperature
$42xx_{16}$	CANopen device temperature
$50xx_{16}$	CANopen device hardware
$60xx_{16}$	CANopen device software
$61xx_{16}$	Internal software
$62xx_{16}$	User software
$63xx_{16}$	Data set
$70xx_{16}$	Additional modules
$80xx_{16}$	Monitoring
$81xx_{16}$	Communication
$82xx_{16}$	Protocol
$90xx_{16}$	External
$F0xx_{16}$	Additional functions
$FFxx_{16}$	CANopen device specific

16-bit error code classes

PRE-DEFINED CONNECTION SET

The CANopen system designer must ensure that unique CAN-IDs are assigned to each CANopen device. In order to simplify this task, CANopen specifies a pre-defined connection set. This is a set of default CAN-IDs assigned to the provided CANopen application layer protocols. Therefore, the default CAN-IDs have a 4-bit function code and a 7-bit node-ID.

The CAN-IDs for the NMT message, the two default Server SDO messages, and the Heartbeat message are not configurable. This is to avoid losing nodes. In other words, there is always the opportunity to access the CANopen device depending on its node-ID (default SDO channel) and observe it (Heartbeat). All other CAN-IDs are a matter of configuration, in particular those of the PDOs. The configuration is done by means of the COB-ID parameters in the related object dictionary entries. COB-ID parameters are 32-bit values. The Bit 29 indicates if 11-bit (default) or 29-bit CAN-IDs are used. The Bit 31 and Bit 30 are used protocol-specific.

A unique node-ID still needs to be assigned by the system designer. There are different options for setting the node-ID. Common ones include DIP switch, local I/O, and the Layer Setting Services as defined in CiA 305. Alternatively, geographical addressing by means of connectors with additional pins can be used to assign a node-ID (e.g. pre-installed wiring harnesses).

CANopen's pre-defined connection set guarantees that no CAN-ID is assigned to two nodes, if the node-IDs are distributed uniquely. The corresponding client SDOs, TIME publisher, and SYNC publisher do not need to be implemented in one entity, they could be in different nodes. The corresponding EMCY consumers and RPDOs can be configured in multiple nodes. The corresponding TPDOs shall be provided by one entity, which could be in different CANopen devices.

For clarity: In many cases, the NMT master device implements all corresponding client SDOs, the SYNC and TIME producers. In a strict master/slave system, the corresponding TPDOs and RPDOs are also hosted in the NMT master device. Additionally, this device consumes all Heartbeat and Boot-up messages.

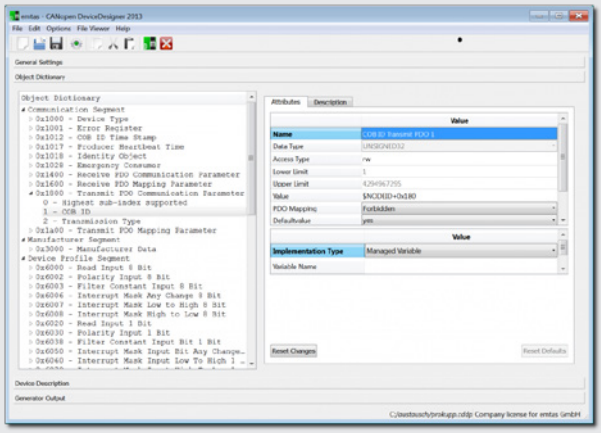
Message	Func. code	Resulting CAN-ID (depending on assigned ID)	Index/sub-index of COB-ID parameter
NMT	0000_2	000_{16} (0)	Not configurable
SYNC	0001_2	080_{16} (128)	$1005\ 00_{16}$
EMCY	0001_2	081_{16} (129) to $0FF_{16}$ (255)	$1014\ 00_{16}$
TIME	0010_2	100_{16} (256)	$1012\ 00_{16}$
TPDO_1	0011_2	181_{16} (385) to $1FF_{16}$ (511)	$1800\ 01_{16}$
RPDO_1	0100_2	201_{16} (513) to $27F_{16}$ (639)	$1400\ 01_{16}$
TPDO_2	0101_2	281_{16} (641) to $2FF_{16}$ (767)	$1801\ 01_{16}$
RPDO_2	0110_2	301_{16} (769) to $37F_{16}$ (895)	$1401\ 01_{16}$
TPDO_3	0111_2	381_{16} (897) to $3FF_{16}$ (1023)	$1802\ 01_{16}$
RPDO_3	1000_2	401_{16} (1025) to $47F_{16}$ (1151)	$1402\ 01_{16}$
TPDO_4	1001_2	481_{16} (1153) to $4FF_{16}$ (1279)	$1803\ 01_{16}$
RPDO_4	1001_2	501_{16} (1281) to $57F_{16}$ (1407)	$1403\ 01_{16}$
TSDO_1 [a]	1011_2	581_{16} (1409) to $5FF_{16}$ (1535)	Not configurable
RSDO_1 [b]	1100_2	601_{16} (1537) to $57F_{16}$ (1663)	Not configurable
Boot-up/ Heartbeat	1110_2	701_{16} (1793) to $77F_{16}$ (1919)	Not configurable

Pre-defined CAN-IDs for CANopen protocols
a = default server SDO message (server-to-client);
b = default server SDO message (client-to-server)

DEVICE AND SYSTEM CONFIGURATION

CANopen devices need to be configured or programmed. In many applications, the NMT master device implements the corresponding SDO clients to the default SDO servers. In a simple master/slave system this NMT master device receives all PDOs from the CANopen NMT slave devices and transmits all PDOs to be received by the CANopen NMT slave devices. No PDO cross communication is pre-defined, except if the CANopen network is based on a CANopen application profile. Besides the configuration of communication parameter, most of the CANopen devices need to be configured, because they provide a default application function and many options.

In general, there are two configuration strategies. The system designer configures all CANopen devices according to the system requirements and stores the configuration in the non-volatile memory. This is supported by means of the store configuration parameter (index 1010_{16}). It provides several options, including which parameters are stored permanently. Of course, there is also a restore parameter (index 1011_{16}), which allows resetting the CANopen device back to factory settings. This configuration approach can be performed with the support of external tools, such as the "emotas CANopen DeviceDesigner".



The CANopen DeviceDesigner is an easy-to-use tool for fast and cost-saving design of CANopen devices.

The other configuration option is to configure all connected CANopen slave devices whenever the system is started. The advantage of this is that even in case of a device substitution, the system works as programmed. On the other hand, you depend on a single entity, the NMT master device, which needs to store the configurations of all CANopen NMT slaves.

Number	Name
CiA 401*	Profile for I/O devices
CiA 402 *	Profile for drives and motion control
CiA 404*	Profile for measuring devices and closed-loop controllers
CiA 406	Profile for encoders
CiA 408	Profile for fluid power technology, proportional valves and hydrostatic transmissions
CiA 410	Profile for inclinometers
CiA 412*	Profiles for medical devices
CiA 413*	Profile for truck gateways
CiA 414*	Profile for weaving machines
CiA 418	Profile for battery modules
CiA 419	Profile for battery chargers
CiA 442	Profile for IEC 61915-2 compatible motor starters
CiA 444*	Profile for spreaders
CiA 445	Profile for RFID devices
CiA 446	Profile for AS-interface gateways
CiA 450	Profile for pumps
CiA 452	Profile for PLCopen motion control
CiA 453	Profile for power supply
CiA 456	Profile for configurable network components
CiA 457	Profile for wireless transmission media-based devices
CiA 458	Profile for energy measurements
CiA 459*	Profile for on-board weighing devices
CiA 460	Profile for service robot controller
CiA 461*	Profile for weighing devices
CiA 462	Profile for item detection devices

CiA device profiles

Number	Name
CiA 415	Profile for sensor systems in road construction and earth moving machines
CiA 416*	Profile for building door control
CiA 417*	Profile for lift control systems
CiA 420*	Profile for extruder downstream devices
CiA 421*	Profile for train control networks
CiA 422*	Profile for municipal vehicles
CiA 423*	Profiles for rail vehicle power drive systems
CiA 424*	Profile for rail vehicle door control systems
CiA 425*	Profile for medical diagnostic add-on modules (e.g. injector)
CiA 426*	Profile for rail vehicle exterior lighting control
CiA 430*	Profile for rail vehicle auxiliary operating systems
CiA 433*	Profile for rail vehicle interior lighting control
CiA 434*	Profile for laboratory automation systems
CiA 436-1	Profile for construction machines
CiA 437*	Profile for grid-based photovoltaic systems
CiA 443	Profile for SIIS level-2 (sub-sea) devices
CiA 447*	Profile for special-purpose car add-on devices
CiA 454*	Profile for energy management systems
CiA 455	Profile for drilling machines

CiA application profiles
** Multi-part specification*

DEVICE AND APPLICATION PROFILES

CiA device profiles specify the CANopen interface of a specific class of devices. Most often implemented is the CiA 401 profile for modular I/O devices. Another profile is CiA 402, which specifies the application behavior of frequency inverters as well as servo- and stepper-motor controllers. It is internationally standardized in the IEC 61800-7 series. These device profiles specify the process data, configuration parameters, and the diagnostic information as well as PDOs.

CiA application profiles specify the entire network. A typical example is the CiA 417 application profile for lift/elevator control systems. It describes all functions as virtual devices. It is possible to implement multiple virtual devices in a single CANopen device, but a virtual device cannot be distributed to several CANopen devices.
In some cases, a set of device profiles achieves the same as an application profile. A typical example is the CiA 420 set of profiles for extruder downstream devices. Application profiles provide an off-the-shelf plug-and-play functionality for the default behavior. Options need to be configured as in the device profile approach.

CANopen FD basics

CANOPEN FD APPLICATION LAYER BASICS

The CANopen FD (Flexible Data-Rate) application layer is specified in CiA 1301, which describes the basic communication services and protocols mapped to the CAN FD data link layer.

The specified communication services and the related protocols (given in brackets) comprise:

- Network management (NMT and Heartbeat)
- Device configuration (USDO)
- Real-time transmission of process data (PDO)
- Node synchronization (SYNC and TIME)
- Diagnostics (EMCY and USDO)

These protocols are mapped by default to CAN data frames in FBFF (FD Base Frame Format). This frame format uses the 11-bit CAN-ID and provides a payload (data field) of up to 64 bytes. The use of 29-bit CAN-IDs is optional (FEFF). The extended payload length compared with the Classical CAN data frames in CBFF (Classical Base Frame Format) and CEFF (Classical Extended Frame Format) is used to improve the protocol functions, especially for USDO and EMCY. Of course, the PDOs also benefit from the longer data field and can now transport up to 64 bytes of process data.

CANopen FD requires the implementation of an object dictionary. This is a list of all parameters representing the CANopen FD device functionality, including process data, configuration options, and diagnostic information. This object dictionary is well structured, and each parameter is addressable by means of a 16-bit index and an 8-bit sub-index. This 24-bit address is used by some communication services as a multiplexer to overcome the limitation of the 2048 identifiers provided by the CAN data link layer. This is so-to-say, a prolongation of the 11-bit CAN-ID used by default for all CANopen protocols. NOTE: The CANopen FD object dictionary is nearly the same as the Classic CANopen object dictionary. Just the error history is new.

Object dictionary structure

The 16-bit index range is structured as follows:

0000_{16}	Reserved
0001_{16} to $025F_{16}$	Data type parameters
0260_{16} to $0FFF_{16}$	Reserved
1000_{16} to $1FFF_{16}$	Communication parameters
2000_{16} to $5FFF_{16}$	Manufacturer-specific parameters
6000_{16} to $67FF_{16}$	Logic device 1 parameters
6800_{16} to $6FFF_{16}$	Logic device 2 parameters
7000_{16} to $77FF_{16}$	Logic device 3 parameters
7800_{16} to $7FFF_{16}$	Logic device 4 parameters
8000_{16} to $87FF_{16}$	Logic device 5 parameters
8800_{16} to $8FFF_{16}$	Logic device 6 parameters
9000_{16} to $97FF_{16}$	Logic device 7 parameters
9800_{16} to $9FFF_{16}$	Logic device 8 parameters
$A000_{16}$ to $AFFF_{16}$	Network variables
$B000_{16}$ to $BFFF_{16}$	System variables
$C000_{16}$ to $FFFF_{16}$	Reserved

Each of these parameters can have up to 256 sub-parameters addressable by means of the 8-bit sub-index. The 00_{16} sub-parameter is used to indicate the highest implemented sub-parameter in case of arrays (all sub-parameters are of the same data type) or records (sub-parameters are of differ-

ent data types). Parameters specified as variables support just the sub-parameter 00_{16}.
The CANopen FD object dictionary supports up to eight logical devices, meaning you can implement multiple device profiles in a single CANopen FD device. This could be a motion controller with eight motor instances or a motion controller with additional I/O functionality.

List of abbreviations

CAN FD	Controller area network with flexible data rate
CBFF	Classical Base Frame Format
CEFF	Classical Extended Frame Format
FBFF	FD Base Frame Format
FEFF	FD Extended Frame Format
ID	Identifier
I/O	Input/output
LSS	Layer setting services
NMT	Network management
PDO	Process data object
SDO	Service data object
USDO	Universal service data object
SYNC	Synchronization object
TIME	Network time object

CANOPEN FD AND THE OSI REFERENCE MODEL

The Open System Interconnection (OSI) reference model from ISO specifies seven layers. The CANopen FD application layer and communication profile, as specified in CiA 1301, mainly covers network, the transport, presentation, and the application layers. The session layer is not used. The data link layer complies with ISO 11898-1 and uses data frames in FBFF (Classical Base Frame Format) by default and optionally data frames in FEFF (Classical Extended Frame Format). Remote frames are allowed, but not recommended at all.

CANopen uses the CAN physical layer as defined in ISO 11898-1:2015 (physical signaling sub-layer) and in ISO 11898-2:20_{16} (physical media access sub-layer). This enables bit-rates up to 1 Mbit/s in the arbitration phase, and up to 2 Mbit/s in the data phase. In order to improve interoperability, CiA 1301 recommends a set of bit-rates.

If other bit-times are required, the CiA 601-3 recommendation should be consulted. It is recommended to use daisy-chain or line topologies with very short stubs. The Bus ends of the network cable need to be terminated by resistors (nominally 120 Ohm each). The physical layer network design at bit-rates above 1 Mbit/s requires additional attention.

The network length at a given bit-rate depends not only on the configured sample, but also on the cables used and non-terminated stubs. In 1-Mbit/s networks you can achieve about 25 m. Using 500 kbit/s, network lengths of up to 125 m are possible. In 250-m networks, bit-rates of up to 250 kbit/s can be reached. Setting the bit-rate to 125 kbit/s allows up to 500 m and at 50 kbit/s the maximum length is 1 km.

The network designer must assign unique CANopen node-IDs to each connected CANopen device. Additionally, all CANopen nodes must use the very same bit-timing settings in all nodes.

The CANopen profile specifications defining the process data, configuration parameters, and the diagnostic information are above the OSI reference model. This also includes the mapping of process data into PDOs.

CiA 302-7 specifies a network layer. The described protocols allow the access of CANopen devices from another CANopen network segment (remote SDO services). This could be used to configure complex CANopen system comprising several segments, from a single point. Of course, this network layer can be also used for diagnosis purposes (remote EMCY services).

OSI reference model

Application level		User program(s)
Data level		CiA 4XX: Device and application profiles
OSI layers		
	Application layer	CiA 301: NMT, Heartbeat, SDO, PDO, SYNC, EMCY, TIME
	Presentation layer	CiA 301: Data types ans encoding rules
	Session layer	Not applicable
	Transport layer	CiA 301: Segmented SDO
	Network layer	(CiA 302-7: SDO and EMCY routing, PDO bridging)*
	Data link layer	ISO 11898-1
	Physical layer	ISO 11898-2, CiA 301 (bit-timing), CiA 303-1 (cable and connectors)

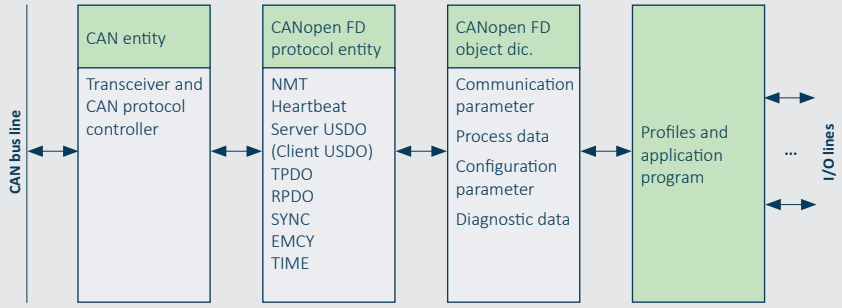
* Only necessary in multiple CANopen FD network architectures

FIELD DEVICE WITH CANOPEN FD COMMUNICATION INTERFACE

The CiA 1301 specification is more precise regarding network and device modeling. The network system model introduces the CiA device profile and the CiA application profile approach. The device profile specifies a single CANopen FD communication interface. The application profile specifies all CANopen FD communication interfaces in a network. These interfaces are virtual, meaning that the one or multiple virtual devices can be implemented in a single CANopen FD device.

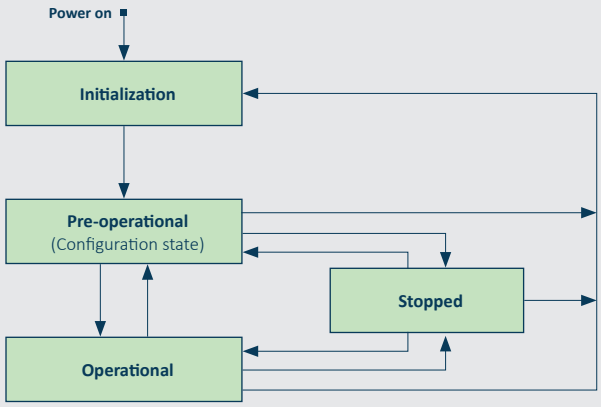
Because CANopen FD integrates a network layer to allow the addressing of CANopen FD devices in other network segments, each CANopen FD devices needs to be addressed by a network-ID plus a node-ID.

The CANopen FD device model shown below includes a CAN entity, which comprises the CAN transceiver and the CAN protocol controller. The CAN protocol controller is, in most cases, on-chip in the micro-controller – sometimes named the host controller. The CANopen FD protocol stack implements the CANopen FD protocols and the CANopen FD object dictionary. And there is the profile and application program of the CANopen FD device, which may comply with one of the CANopen profiles specified by CiA. Of course, the CANopen FD device may implement just manufacturer-specific profiles.



The CANopen FD device model

The CANopen FD device's communication and application parameters – accessible by means of the object dictionary – are represented electronically in the Electronic Data Sheet (EDS). The EDS provides the parameter and sub-parameter attributes. This includes index and sub-index, name, object code, data type, category (or entry category), access, PDO mapping, value range, and default value. EDS are used to teach device and system configuration tools the provided functionality. Such tools can also be implemented in the host controller with NMT master capability.



NMT slave state machine

NETWORK MANAGEMENT (NMT)

CANopen FD network management is based on a master/slave approach. The device with NMT master functionality controls the NMT slave devices. Each NMT slaves can be in one of three static states:

- Pre-operational: All CANopen services can be used, except PDO services,
- Operational: All CANopen services can be used, or
- Stopped: No CANopen services can be used, except NMT and Heartbeat

There are additional temporary states, which the device transmits automatically after power-on and after reset. The state diagram shown provides details of the possible state transitions. Those transitions are commended by the NMT master or device internally by the application.

The NMT commands from the NMT master devices are confirmed on the application program level by means of the Heartbeat message. The NMT command message is a two-byte message containing the command in one byte and the addressed node-ID in the other.
A node-ID of "0" indicates a broadcast command, meaning that all nodes shall perform the command. It is mapped to the CAN FD data frame with CAN-ID "0", which is the highest priority. The above-mentioned confirmation is a one-byte message providing the current state (pre-operational, operational, or stopped) of the related NMT slave device. It is mapped depending on the node-ID of the transmitting device to a CAN FD data frame with a CAN-ID of 800_{16} plus node-ID. The Heartbeat message is sent periodically with the user-configurable heartbeat-producer-time given in the object dictionary.

After power-on, the NMT slave transits automatically into the NMT pre-operational state and waits to be configured and started by the NMT master device. It is also possible to configure an NMT slave device to be self-starting. This is required in some CANopen FD applications. There are two reset commands: one simply resets the communication parameters to the default or the configured values; the other resets the communication and the profile parameters. After the reset procedure, the NMT slave device transitions automatically to the NMT pre-operation state.

The NMT stopped state can be used to stop a device communicating USDO and PDO messages. It just accepts the NMT message and produces the Heartbeat message. In some cases, it is desirable for the CANopen FD device to not transmit any message. To achieve this, the heartbeat-producer-time in the object dictionary needs to be configured to "0".

HEARTBEAT SERVICES

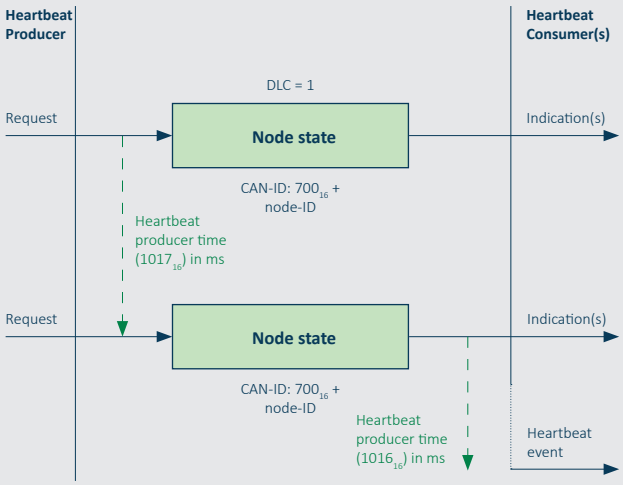
CANopen FD network management uses the Heartbeat message as the confirmation of the NMT command sent by the NMT master device. Additionally, any CANopen FD device can use it to check the availability of any other CANopen FD device. This is necessary in all cases in which CANopen FD devices transmit PDO messages only on state-of-change events. The subscriber of such PDO messages could not know if there is no event or if the device is no longer available. The reception of the Heartbeat message of the related device indicates that it is still alive.

The transmission period is configurable by means of the heartbeat-producer-time parameter in the object dictionary of the transmitting device. The indication that a device is missing can be configured in any interested device, by means of the heartbeat-consumer-time array. As a rule of thumb, the consumer time should be twice that of the producer time. The producer and consumer time configurations are highly application specific.

In some applications, CANopen FD devices cross-check each other by consuming the Heartbeat messages. Sensors normally do not consume Heartbeat messages; they just produce their PDO messages and their Heartbeat messages to respectively indicate that they are still alive to confirm the NMT command.

System designers are responsible for the configuration of the Heartbeat timing. There could be application-specific requirements that a device missing an essential device stops the production of its own Heartbeat message. The system designer should take care that the overall system availability is still sufficient.

The 1-byte Heartbeat message uses a CAN FD data frame with the ID 700_{16} plus producer node-ID. The one-byte content indicates the NMT status. This means, the consumer can also detect that a device has not sent PDO messages because of its NMT state. This can be evaluated in the consumer's application program.



Node state values:
04_{16} (Stopped), 05_{16} (Operational), $7F_{16}$ (Pre-operational)

Heartbeat protocol: When the configured heartbeat-consumer-time elapses, the application gets an indication and may stop the production of its own Heartbeat message, because a missing CANopen FD device could be critical for the functionality.

Heartbeat message content

The Heartbeat message is mapped to a single CAN FD data frame with a 1-byte data field. This byte provides the NMT status of the CANopen FD device. It could be in pre-operational state ($7F_{16}$), operational state (05_{16}), or stopped state (04_{16}). In the pre-operational state, the CANopen FD device does not process PDOs and in the stopped state it only supports the NMT message and sends its Heartbeat. In the operational state all CANopen functions are provided.

If the CAN FD data frame contains 00_{16}, it is interpreted as a Boot-up message indicating that this CANopen FD device has just entered the network. This happens after initial power-on, a power-cycle, and after application and communication reset. The next CAN FD data frame with the same CAN-ID and content unequal 00_{16} is interpreted as Heartbeat message. The NMT master device uses the Heartbeat message as confirmation of its NMT commands.

USDO SERVICES

CiA 1301 specifies several USDO services. The main purpose of all these services is to read from and write data to the CANopen FD object dictionary. The USDO client always has the initiative sending an USDO read or write request. In opposition to the SDO communication in classic CANopen, the USDO client provides the destination information in its protocol. Therefore, the USDO protocols support broadcast, multi-cast, and uni-cast addressing. This means that all CANopen FD nodes can communicate via USDO with each other CANopen FD node. USDO services are intended for configuration and diagnostic tasks. Of course, process data can be transferred, as well.

CANopen FD distinguishes between USDO local services and USDO remote services. The USDO local services are used to communicate in the very same network segment. This is indicated in the USDO protocol by means of the network-ID. USDO remote services are intended for communication with CANopen FD devices in another segment with another network-ID.

USDO services are always confirmed. Read requests are confirmed by the USDO data received from the USDO server. Write requests are confirmed by means of USDO server segments indicating that the data is stored in the CANopen FD object dictionary. The USDO client needs to support an internal USDO client-response timeout. The USDO client-response timeout indicates the duration in which the USDO client expects all responses to its request. The USDO client does not consider responses that are received after the timeout-time has elapsed.

There are specified USDO expedited and segmented services. The expedited ones are mapped to just one packet, because the payload can carry all application data. The USDO segmented services are mapped to multiple USDO protocol packets. They are used to up- or download large amounts of data. Each packet is confirmed, which consumes some bandwidth. Therefore, CANopen FD specifies also so-called USDO bulk services. In these services not every packet is confirmed individually, but instead in a group of packets.

If the USDO service cannot be handled correctly, either the USDO server or the USDO client sends an USDO abort message. The USDO abort message provides a 4-byte abort code, giving the user information about the reason.

USDO services provide pre-configured broad-, multi-, and uni-cast communication, which unburdens the system designer from assigning CAN-IDs. This means that it is possible to run a network application using only USDO services. Of course, Heartbeat services and EMCY services are needed, too.

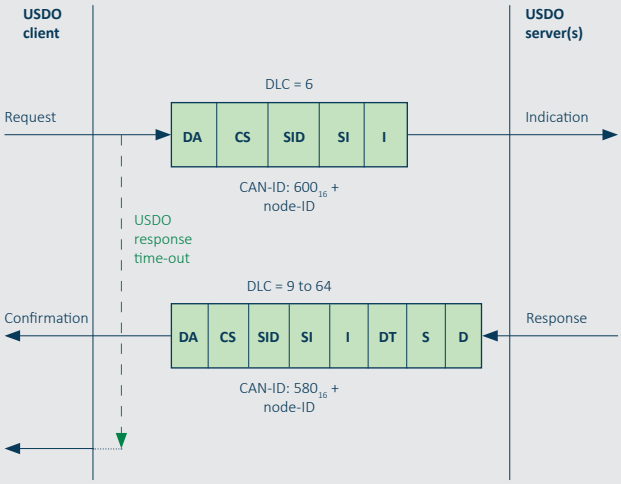
The current USDO service specification allows multiple SDO sessions. Therefore, the USDO protocol contains a session-ID. The data type indication is also new.

The next edition of CiA 1301 plans to support transmitting multiple object dictionary entries by means of one single USDO service. This could be used, for example, for PDO mapping or communication parameters, or any other selection of multiple object dictionary entries.

USDO protocol examples

USDO messages have a protocol overhead of different lengths depending on the related USDO service. This overhead contains, for example, the kind of service (command specifier) and the 24-bit address of the object dictionary parameter to be written or to be read (index and sub-index). The protocol details are specified in CiA 1301. The table shown provides the command specifiers of the defined USDO protocols for local and remote network accesses.

In general, there are protocols for non-segmented USDO messages. Those do not require more than the 64-byte payload provided by the CAN FD data link layer. If the USDO service needs more payload, the CANopen FD protocol stack segments the data to be transmitted, and then the CANopen FD protocol stack of the receiving nodes re-assemble them.

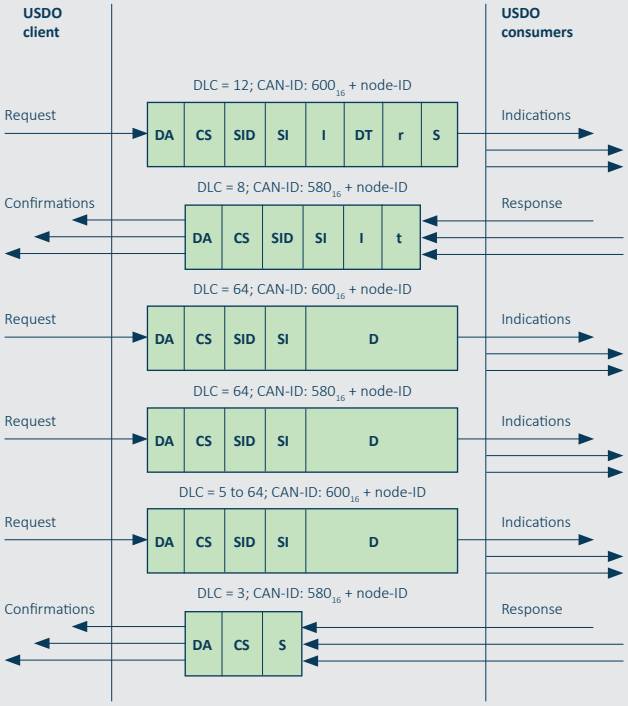


Not segmented local uni-cast USDO protocol sequence.

CS = command specifier; D = application data to be uploaded; DA = destination address; DT = data type; I = index; SI = sub-index; SID = session-ID

Code (local)	Code (remote)	Protocol name
01_h	81_h	Download expedited request
02_h	82_h	Download segmented initial request
03_h	83_h	Download segmented intermediate request (segment)
04_h	84_h	Download segmented last request (end)
05_h	85_h	Reserved for future use by CiA
06_h	86_h	Download bulk transfer initial request
07_h	87_h	Download bulk transfer segmented
08_h	88_h	Download bulk transfer final
09_h to 10_h	89_h to 90_h	Reserved for future use by CiA
11_h	91_h	Upload request
12_h	92_h	Reserved for future use by CiA
13_h	93_h	Upload segmented intermediate request (segment)
14_h to 20_h	94_h to $A0_h$	Reserved for future use by CiA
21_h	$A1_h$	Download expedited response
22_h	$A2_h$	Download segmented initial response
23_h	$A3_h$	Download segmented intermediate response (segment)
24_h	$A4_h$	Download segmented last response (end)
25_h	$A5_h$	Reserved for future use by CiA
26_h	$A6_h$	Download bulk transfer initial response
27_h	$A7_h$	Reserved for compatibility reasons
28_h	$A8_h$	Download bulk transfer final response
29_h to 30_h	$A9_h$ to $B0_h$	Reserved for future use by CiA
31_h	$B1_h$	Upload expedited response
32_h	$B2_h$	Upload segmented initial response
33_h	$B3_h$	Upload segmented intermediate response (segment)
34_h	$B4_h$	Upload segmented last response (end)
35_h to $7E_h$	$B5_h$ to FE_h	Reserved for future use by CiA
$7F_h$	FF_h	USDO abort service

USDO command specifiers



USDO local download bulk transfer broadcast protocol sequence.

CS = command specifier; D = application data segment to be downloaded; DA = destination address; DT = data type; I = index; r = reserved; S = total size of data to be downloaded; SI = sub-index; SID = session-ID; t = time for processing segments

PDO SERVICES

In CANopen FD, the Process Data Object (PDO) functionality is nearly the same as that used in Classic CANopen. PDOs are used to transmit process data, which could be for measured values, or commands, most of which are time-critical. The PDO service is not confirmed, it is a producer/consumer service or in other words a publisher/subscriber service.

PDO messages in CANopen FD contain up to 64 bytes of process data. They are mapped to a single CAN FD data frame using a CAN-ID determined by the COB-ID parameter given in the CANopen FD object dictionary. This parameter is part of the PDO communication parameter set. This parameter set also comprises the transmission type, which indicates the kind of transmission mode respectively reception mode.

In general, CANopen FD allows the transmission of PDOs asynchronously and synchronously. The triggering of asynchronous PDOs depends on device-internal events. This can be the change of process data transmitted in this PDO or the elapsing of the event-timer – also part of the PDO communication parameter set. The event-timer enables periodical transmission of the PDO. But bear in mind, the event-timer is a local timer and not synchronized with the other CANopen FD device in the network.

The synchronous transmission of PDOs is a unique feature of the CANopen FD application layer. It is based on the periodical transmission of the SYNC message, which uses a single CAN data frame with the default high-priority CAN-ID of 80_{16}. The SYNC message triggers the transmission of all synchronous PDOs in the CANopen FD network. This means, all sensors measure their values at the very same moment. Depending on the internal delay and the priority of the PDO message, these values are transmitted one after the other, but all have been measured simultaneously.

In order to synchronize “fast” and “slow” CANopen FD devices, the synchronous TPDOs can be configured to serve each, each second up to each 240th SYNC message. But with mechanism you cannot group synchronous PDOs. For this feature, you need to use the SYNC message with a 1-byte counter. Additionally, the SYNC start value (sub-index 06_{16}) needs to be configured. Now, you can start one group of synchronous PDOs, for example, at the SYNC counter value of 1 serving each second SYNC message, and another group at the SYNC counter value of 2 also serving each second SYNC message. Of course, you can configure more complex behavior to achieve a stable average busload.

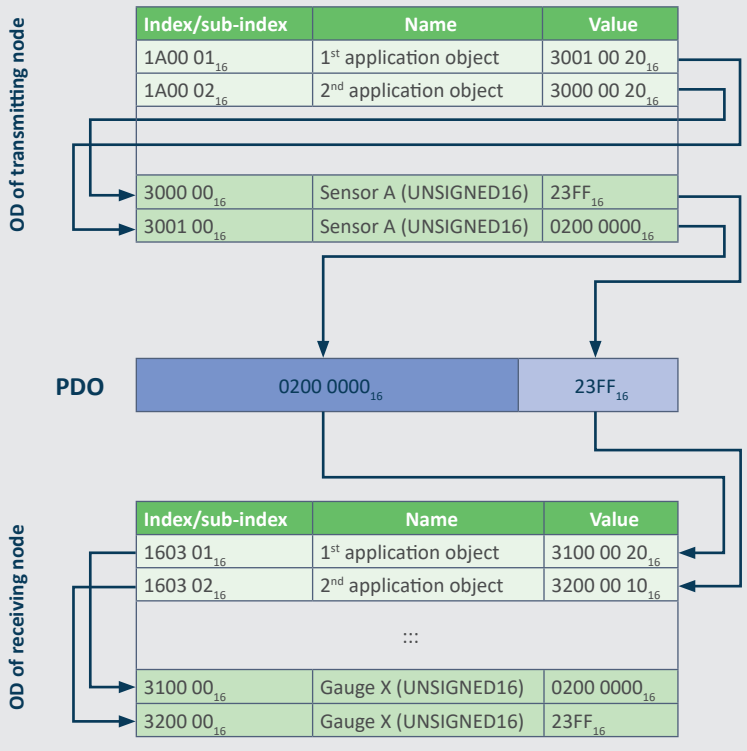
In order to synchronize outputs in different devices, the related RPDOs need to be configured as synchronous. This means, the received commands are not performed immediately, but when the next SYNC message is received. At this time all devices perform synchronized the requested actuation.

Another unique feature is the PDO inhibit time. This configurable time circumvents the transmission of the PDO for the configured period. This means, a high-priority PDO cannot occupy the entire bandwidth, and lower-prior PDOs can win bus-arbitration. With this feature you can design a predictable transmission of Change-of-State (CoS) triggered PDOs.

There is a second PDO parameter set specifying the content of the PDO message. It is possible to map multiple process data into one PDO message. Bit-wise mapping is supported, but not recommended. The related PDO mapping parameter array in the object dictionary provides the object dictionary addresses (16-bit index plus 8-bit sub-index), where the process data value is stored (in case of a Transmit-PDO) or should be stored (in case of a Receive-PDO).

The PDO mapping parameters could be different on the transmitting and the receiving side, the example shows a sensor device transmitting two process data and a display device with virtual gauges receiving the PDO.
The system designer has several opportunities to optimize the PDO communication to the application requirements:

- PDO prioritization: Configuring the COB-ID parameter with an appropriate CAN-ID value changes the prior when accessing the CAN network.
- PDO linking: Configuring the COB-ID parameter of the RPDO to a dedicated CAN-ID value is a subscription of the corresponding TPDO. It is possible that multiple or all CANopen devices subscribe to the same TPDO (multi-cast and broadcast communication)
- PDO scheduling: Configuring the transmission type sub-parameter of a TPDO determines the sending behavior (e.g. change-of-state (CoS), periodically, or synchronously). Configuring the transmission type parameter of a RPDO determines the reception behavior (performing the required action immediately or with the reception of the next SYNC message).
- PDO mapping: Configuring the PDO mapping parameter set determines which process data are mapped into the TPDO or where to enter the received process data (RPDO).



The PDO mapping parameters on the transmitting and the receiving side can differ, the example shows a sensor device transmitting two process data and a display device with virtual gauges receiving the PDO.

Transmission type values

The transmission type sub-parameter determines the scheduling of TPDO. In the case of a synchronous transmission, the TPDO can be configured to serve each received SYNC message, each second, etc. Remotely requested TPDO should not be used, due to a completely specified behavior (the reaction on CAN remote frames can be implemented differently, for example).

Value	Description
00_{16}	Acyclic synchronous: Triggered when the SYNC message is received and one of the mapped process data has changed its value after the last transmission.
01_{16} to $F0_{16}$	Cyclic synchronous: Triggered when the SYNC message is received (01_{16}), each second SYNC message is received (02_{16}), etc. This supports fast and slow changing process data.
$F1_{16}$ to FB_{16}	Reserved
FC_{16}	Synchronous RTR only: not recommended anymore
FD_{16}	Asynchronous RTR only: not recommended anymore
FE_{16}	Asynchronous: Triggered by an internal event (e.g. change-of-state of one of the mapped process data or elapsing of the event-timer or any other event). The device manufacturer specifies the internal event triggering the TPDO transmission.
FF_{16}	Asynchronous: As before, but the CiA profile specifies the internal event triggering the TDPO transmission.

Transmission type values

PDO communication parameter set

This parameter set is a record. The first sub-parameter (highest sub-index supported) indicates the sub-parameter with highest number. The COB-ID sub-parameter contains the CAN-ID to be used and some additional protocol control bits. The transmission type sub-parameter determines the triggering (TPDO) or reception (RPDO) behavior. The inhibit-time sub-parameter provides the time in milliseconds, before the TPDO is allowed to send it again. The event timer sub-parameter specifies the period in milliseconds of the TPDO transmission respectively the time-out (missing) of a RPDO. The SYNC start value sub-parameter indicates when the synchronous TPDO is transmitted first. The value of this sub-parameter needs to match with the value given in the 1-byte SYNC message.

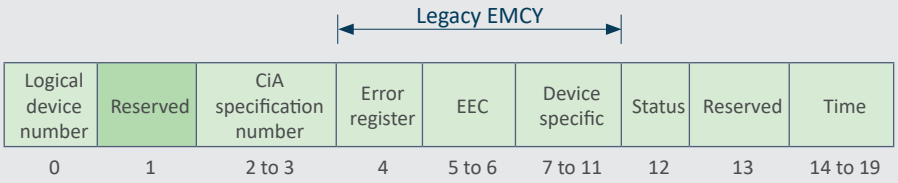
Sub-index	Name	Data type
00_{16}	Highest sub-index sup.	Unsigned8 (BYTE)
01_{16}	COB-ID	Unsigned32 (DWORD)
02_{16}	Transmission type	Unsigned8 (BYTE)
03_{16}	Inhibit-time	Unsigned16 (WORD)
04_{16}	Reserved	Not applicable
05_{16}	Event timer	Unsigned16 (WORD)
06_{16}	SYNC start value	Unsigned8 (BYTE)

PDO communication parameter set

EMCY SERVICE

The EMCY service provides information about the error status of the CANopen device in respect to the communication and the application. It is mapped to a producer/consumer protocol. The producer sends the EMCY message indicating some detected “errors”.

The 20-byte message contains the same information as the EMCY message of the Classic CANopen application layer (Legacy EMCY): the one-byte error register, the two-byte emergency error code, and five bytes, which are CANopen profile-specific. Some profiles allow using the five bytes for manufacturer-specific purposes.
In addition, the CANopen FD EMCY message provides the related logical device number, the related CiA specification number, the status of the detected error, and the time of the error occurrence (TIME-OF-DAY). The type of error (status) indicates if the error is recoverable or not. Warnings may also be indicated.
The error register is a variable listed in the CANopen object dictionary. It controls the Emergency state machine. If any “error” is detected, the CANopen device triggers the EMCY service and sends the EMCY message. If further “errors” occur, the CANopen device remains in the error state. Only if all “errors” are recovered, will the CANopen device transition into the error-free state. The error state



Structure of the EMCY protocol: New are the logical device number, the CiA specification number, the status, and the time

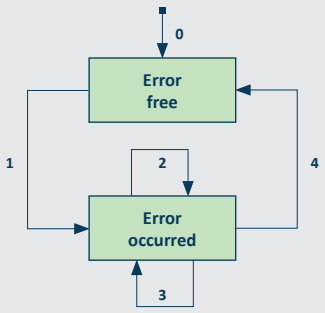
behavior could be defined by CANopen profiles. In the CiA 401 profile for modular I/O devices, the outputs are set to pre-defined values when the device is in error state.

The system designer needs to configure the EMCY message consumers depending on the application requirements. This is achieved by means of the emergency consumer parameter listed in the CANopen object dictionary. This array parameter contains all COB-IDs of EMCY messages, which are consumed. NOTE: The COB-ID is a 32-bit value providing the CAN-ID of the data frames to be consumed as well as three control bits.

The reaction on the received EMCY messages is highly application-specific. By default, the EMCY message is transmitted in a single CAN FD data frame with a high-priority CAN ID: 80_{16} plus node-ID. The system designer assigns a unique node-ID to the CANopen devices. The EMCY COB-ID parameter, which contains the CAN-ID to be used can be configured by means of an SDO write service to the CANopen object dictionary.

Value	Description
$00xx_{16}$	No error or reset
$10xx_{16}$	Generic error
$20xx_{16}$	Current
$21xx_{16}$	Current, CANopen FD device input side
$22xx_{16}$	Current inside the CANopen FD device
$23xx_{16}$	Current, CANopen FD device output side
$30xx_{16}$	Voltage
$31xx_{16}$	Mains
$32xx_{16}$	Voltage inside the CANopen FD device
$33xx_{16}$	Output voltage
$40xx_{16}$	Temperature
$41xx_{16}$	Ambient temperature
$42xx_{16}$	CANopen FD device temperature
$50xx_{16}$	CANopen FD device hardware
$60xx_{16}$	CANopen FD device software
$61xx_{16}$	Internal software
$62xx_{16}$	User software
$63xx_{16}$	Data set
$70xx_{16}$	Additional modules
$80xx_{16}$	Monitoring
$81xx_{16}$	Communication
$82xx_{16}$	Protocol
$90xx_{16}$	External
$F0xx_{16}$	Additional functions
$FFxx_{16}$	CANopen FD device specific

16-bit error code classes



KEY
0: No error detected
1: First error detected
2: One error resolution, but not all
3: Additional error detected
4: All errors resolved

Error state machine

Communication errors

CiA 1301 specifies some detailed communication error codes. They are related to the CAN FD data link layer protocol as well as to the CANopen FD application layer protocols. The implementation of the following error codes is recommended:

- 8110_{16}: CAN overrun (data frame lost)
- 8120_{16}: CAN protocol controller in error passive mode (node cannot indicate bus errors, when it is a receiver)
- 8140_{16}: CAN node recovered from bus-off (may have missed some CAN data frames)
- 8150_{16}: CAN-ID collision (node has detected data frames with CAN-IDs that are assigned to it)

Support of the following CANopen FD protocol error codes is also recommended:

- 8130_{16}: Heartbeat error (one of the supervised CANopen device has not sent its Heartbeat)
- 8160_{16}: USDO source client or source server number collision (this indicates a double assignment of node-IDs)
- 8210_{16}: PDO not processed due to length error (PDO receiving CANopen device detected a mismatch of the PDO length)
- 8220_{16}: PDO length exceeded (the mapped process data is longer than 8 bytes)
- 8230_{16}: DAM MPDO nor processed, destination element not available
- 8240_{16}: Unexpected SYNC message length (SYNC consumer expect another SYNC format with or without SYNC counter)
- 8250_{16}: RPDO timeout (RPDO event timer elapsed)
- $8F01_{16}$ to $8F7F_{16}$: Heartbeat event caused by node-ID 1 to node-ID 127

PRE-DEFINED CONNECTION SET

The CANopen FD system designer must assign unique CAN-IDs to the CANopen FD devices. In order to simplify this task, CANopen specifies a pre-defined connection set. This is a set of default CAN-IDs assigned to the provided CANopen FD application layer protocols. Therefore, the default CAN-IDs have a 4-bit function code and the 7-bit node-ID.

Message	Func. code	Resulting CAN-ID (depending on assigned ID)	Index/sub-index of COB-ID parameter
NMT	0000_2	000_{16} (0)	Not configurable
SYNC	0001_2	080_{16} (128)	$1005\ 00_{16}$
EMCY	0001_2	081_{16} (129) to $0FF_{16}$ (255)	$1014\ 00_{16}$
TIME	0010_2	100_{16} (256)	$1012\ 00_{16}$
TPDO_1	0011_2	181_{16} (385) to $1FF_{16}$ (511)	$1800\ 01_{16}$
RPDO_1	0100_2	201_{16} (513) to $27F_{16}$ (639)	$1400\ 01_{16}$
TPDO_2	0101_2	281_{16} (641) to $2FF_{16}$ (767)	$1801\ 01_{16}$
RPDO_2	0110_2	301_{16} (769) to $37F_{16}$ (895)	$1401\ 01_{16}$
TPDO_3	0111_2	381_{16} (897) to $3FF_{16}$ (1023)	$1802\ 01_{16}$
RPDO_3	1000_2	401_{16} (1025) to $47F_{16}$ (1151)	$1402\ 01_{16}$
TPDO_4	1001_2	481_{16} (1153) to $4FF_{16}$ (1279)	$1803\ 01_{16}$
RPDO_4	1001_2	501_{16} (1281) to $57F_{16}$ (1407)	$1403\ 01_{16}$
TSDO_1 [a]	1011_2	581_{16} (1409) to $5FF_{16}$ (1535)	Not configurable
RSDO_1 [b]	1100_2	601_{16} (1537) to $57F_{16}$ (1663)	Not configurable
Boot-up/ Heartbeat	1110_2	701_{16} (1793) to $77F_{16}$ (1919)	Not configurable

Pre-defined CAN-IDs for CANopen protocols
a = default server SDO message (server-to-client);
b = default server SDO message (client-to-server)

The CAN-IDs for the NMT message, the USDO client and USDO server messages, and the Heartbeat message are not configurable. This is to avoid losing nodes. All other CAN-IDs are a matter of configuration, in particular those of the PDOs. The configuration is performed by means of the COB-ID parameters in the related object dictionary entries. COB-ID parameters are 32-bit values. The Bit 29 indicates if 11-bit (default) or 29-bit CAN-IDs are used. The Bit 31 and Bit 30 are protocol-specific.

Of course, the node-ID still needs to be assigned uniquely by the system designer. There are different options to set the node-ID. Common are DIP switch, local I/O, and the Layer Setting Services as defined in CiA 1305 (under development). Alternatively, geographical addressing by means of connectors with additional pins can be used to assign a node-ID (e.g. pre-installed wiring harnesses).

The CANopen FD pre-defined connection set guarantees that no CAN-ID is assigned to two nodes, if the node-IDs are distributed uniquely. The TIME publisher, and SYNC publisher do not need to be implemented in one entity, they could be in different nodes. The corresponding EMCY consumers and RPDOs can be configured in multiple nodes. The RPDOs and corresponding TPDOs shall be provided by just one entity, which could be in different CANopen FD devices.

Each CANopen FD device implements an USDO client as well as USDO servers for all other CANopen FD devices in the network. This means, a full-meshed USDO communication is pre-defined.

DEVICE AND SYSTEM CONFIGURATION

CANopen FD devices need to be configured or programmed. In a simple master/slave system this NMT master device receives all PDOs from the CANopen FD devices and transmits all PDOs to be received by the CANopen FD devices. No PDO cross communication is pre-defined, except the CANopen network is based on a CANopen application profile. Besides the configuration of communication parameter, most of the CANopen FD devices need to be configured, because they provide a default application function and many options.

In general, there are two configuration strategies. The system designer configures all CANopen FD devices according to the system requirements and stores the configuration in the non-volatile memory. This is supported by means of the store-configuration parameter (index 1010_{16}). It provides several options, including which parameters are stored permanently. There is also a restore parameter (index 1011_{16}), which allows the CANopen FD device to be reset back to its factory settings. This configuration approach can be achieved with the support of external tools.

The other configuration option is to configure all connected CANopen FD devices whenever the system is started. The advantage of this is the event of a device substitution, the system works as programmed. On the other hand, you depend on a single entity, the NMT master device, which needs to store the configurations of all CANopen FD devices.

DEVICE AND APPLICATION PROFILES

CiA profiles can be used for Classic CANopen and CANopen FD. Of course, if the larger PDO payload of CANopen FD is used, additional PDO mapping specification is required to achieve the same level of interoperability as in Classic CANopen. The CiA profiles will be updated step-by-step to standardize the PDO mapping beyond the 8-byte limit.

The larger payload can also be used to provide additional functional safety and cyber security protocols. This would allow end-to-end protection, which is not yet standardized by CiA.

Number	Name
CiA 401*	Profile for I/O devices
CiA 402 *	Profile for drives and motion control
CiA 404*	Profile for measuring devices and closed-loop controllers
CiA 406	Profile for encoders
CiA 408	Profile for fluid power technology, proportional valves and hydrostatic transmissions
CiA 410	Profile for inclinometers
CiA 412*	Profiles for medical devices
CiA 413*	Profile for truck gateways
CiA 414*	Profile for weaving machines
CiA 418	Profile for battery modules
CiA 419	Profile for battery chargers
CiA 442	Profile for IEC 61915-2 compatible motor starters
CiA 444*	Profile for spreaders
CiA 445	Profile for RFID devices
CiA 446	Profile for AS-interface gateways
CiA 450	Profile for pumps
CiA 452	Profile for PLCopen motion control
CiA 453	Profile for power supply
CiA 456	Profile for configurable network components
CiA 457	Profile for wireless transmission media-based devices
CiA 458	Profile for energy measurements
CiA 459*	Profile for on-board weighing devices
CiA 460	Profile for service robot controller
CiA 461*	Profile for weighing devices
CiA 462	Profile for item detection devices

CiA device profiles

Number	Name
CiA 415	Profile for sensor systems in road construction and earth moving machines
CiA 416*	Profile for building door control
CiA 417*	Profile for lift control systems
CiA 420*	Profile for extruder downstream devices
CiA 421*	Profile for train control networks
CiA 422*	Profile for municipal vehicles
CiA 423*	Profiles for rail vehicle power drive systems
CiA 424*	Profile for rail vehicle door control systems
CiA 425*	Profile for medical diagnostic add-on modules (e.g. injector)
CiA 426*	Profile for rail vehicle exterior lighting control
CiA 430*	Profile for rail vehicle auxiliary operating systems
CiA 433*	Profile for rail vehicle interior lighting control
CiA 434*	Profile for laboratory automation systems
CiA 436-1	Profile for construction machines
CiA 437*	Profile for grid-based photovoltaic systems
CiA 443	Profile for SIIS level-2 (sub-sea) devices
CiA 447*	Profile for special-purpose car add-on devices
CiA 454*	Profile for energy management systems
CiA 455	Profile for drilling machines

CiA application profiles

** Multi-part specification*

Implementation of CANopen and CANopen FD

MODULE BASED IMPLEMENTATION

Besides the implementation of CANopen by using a protocol stack, also ready made modules can be implemented on the micro-controller based target system. For this, HMS offers the Anybus CompactCom module, which handles the entire CANopen communication and enables fast and cost effective implementation of a CANopen slave interface for products which are sold in the lower or mid-range volume area.
The Anybus CompactCom solution is available as chip, brick or module, and thus adapts in the best possible way to the respective requirements of the application. Furthermore the multi-network approach of the Anybus CompactCom solutions makes it easy to switch to other protocols just by changing the module and by making minor adaptations at the application. This makes the use of the module extremely future-proof and flexible, as there is no need to commit to a specific protocol.

PROTOCOL SOFTWARE BASED IMPLEMENTATION

For the implementation of CANopen and CANopen FD in embedded devices protocol stacks are can be used, which are usually offered in form of company or site licenses. In cooperation with emotas, HMS offers comprehensively tested and ANSI-C compatible CANopen and CANopen FD protocol stacks.
The stacks are available for "Slave", "Master/Slave" or "Manager" devices and proven in many applications.

To connect the CANopen or CANopen FD stacks to multiple CAN controllers and CPU types, a well-defined driver interface is used. Using this driver interface the stacks can also be easily adapted to new CAN controllers or CPU types. The stacks can be used stand-alone or with various real-time Operating Systems, such as ThreadX, FreeRTOS, Keil RTX or TI-RTOS as well as with Linux (SocketCAN, can4linux), QNX or with real-time extensions for Windows.

What does the implementation of a CANopen stack involve?

The emotas protocol stacks are provided with comprehensive documentation and sample programs. Project files are supplied with the sample programs that allow direct integration into the corresponding development environments of the compiler manufacturer.

The implementation is carried out as follows:

- Adaptation to the hardware of the target platform (Timer, Interrupt system)
- Creation of object dictionary entries
- Adaptation of configuration file
- Compiling and testing

Early CANopen implementations on the target system can be set up within a few days.

Develop or buy?

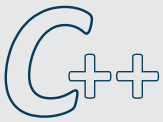
Time and again there are developers who consider whether they should develop the CANopen or CANopen FD protocol software themselves, because the specifications are freely available without license costs and there are also several open-source initiatives to date. This is certainly an interesting development challenge, however, from an economic point of view, it is not at all worthwhile, as mature CANopen and CANopen FD protocol software is offered by several manufacturers at very attractive prices.
For the development of a sufficiently tested and documented protocol stack a development time of at least 4 to 6 months is to be expected, even if the full functional scope of the specifications is not implemented. However, it remains to be seen whether an "initial implementation" is already a sufficiently optimized, fault-free product and whether the "subtleties" of the not particularly straightforward CANopen specification are correctly implemented. The possible setting up of free or open implementations does little to change this fact. Most of these approaches are only really of interest for training or other non-commercial environments.

If we assume that an in-house development of a CANopen protocol stack could be accomplished within four man months, at an hourly rate of EUR 60, the costs for such a development would be more than EUR 35.000. However, this is a very optimistic estimation and a well-tested and reliable solution supplied as source code off the shelf is available for a fraction of this price. Therefore, an in-house development of a CANopen protocol stack is relatively pointless from an economic point of view, including the time-to-market.

Products from HMS for implementing CANopen

Anybus CompactCom Module from HMS

- Ready made modules enable quick implementation of CANopen
- Available as chip, brick or module for the best possible adaptation to the customer application
- Easy switch to other protocols with only little effort, no need to commit to a specific protocol



Protocol stacks from emotas

- Highly flexible way for implementing CANopen and CANopen FD
- Available for a variety of target systems and easily adaptable to further systems
- For master, manager and slave applications
- Continuously maintained
- Ideal for products which are soled in high volumes

Analysis and configuration tools from HMS and emotas

canAnalyser 3

- Analysis tool for CAN and CANopen
- Ideal for device testing during development and system stimulation

emotas DeviceDesigner and -Explorer

- Easy object dictionary creation based on predefined, standardized profiles
- Provides master functionality to allow analysis and configuration

Discover more on www.hms-networks.com