# Configuration Guideline for CANopen Networks

Martin Rostan, Beckhoff

**Unlike most other fieldbus systems, CANopen provides many degrees of freedom to configure the communication behaviour of the network. Product manuals rarely describe simple paths to achieve the optimal configuration. Network analysis tools generate a lot of data that has to be interpreted by CANopen experts. Most CANopen users have difficulties to configure their network smartly and quickly and end up with sub-optimal setups.**

**The paper first briefly explains the various configuration possibilities and shows the optimisation targets, depending on the application requirements. The influence of the parameters on reaction time, determinism and bus load is shown. Then a configuration guideline for CANopen networks is introduced, which allows one to optimize the network step by step using simple indicators.**

## Process Data Objects

In many fieldbus systems the entire process image is continuously transferred - usually in a more or less cyclic manner. CANopen is not limited to this communication principle, since the multi-master bus access protocol allows CAN to offer other methods. The process data in CANopen is divided into segments with a maximum of 8 bytes. These segments are known as process data objects (PDOs). The PDOs each correspond to a CAN telegram, whose specific CAN identifier is used to allocate them and to determine their priority. The PDOs are named from the point of view of the node: receive PDOs (RxPDOs) are received by the node and – in case of an I/O device - contain output data, while transmit PDOs (TxPDOs) are sent by the I/O device and contain input data.

## Mapping

Typically there are several RxPDO and TxPDOs available for the process data exchange. The default allocation of input or output data to these PDOs is called default mapping and is defined in the device profiles. It is recorded in the mapping tables in the object directory of the device. These mapping tables create the cross-reference between the application data in the object directory (e.g. digital and analogue input data) and the sequence in the process data objects. The first location in the mapping table (sub-index 0) contains the number of mapped objects that are listed after it. The mapping tables are located in the object directory at index 0x1600ff for the RxPDOs and at 0x1A00ff for the TxPDOs.

The default mapping of process data objects is usually sufficient to meet the application requirements. For special types of application the mapping can nevertheless be altered: many CANopen devices support variable mapping, in which the application objects can be freely allocated to the PDOs.

## Communication parameters

The PDOs can be given different communication parameters according to the requirements of the application. Like all parameters they can be found in the object dictionary of the device. The parameters are located in the object directory from index 0x1400ff (RxPDOs) and index 0x1800ff (TxPDO). There are entries for up 512 RxPDOs and 512 TxPDOs. For each available PDO the communication parameter entry must be present, but write access can be restricted.

Each entry is a data structure of up to five parameters, which are now explained in more detail:

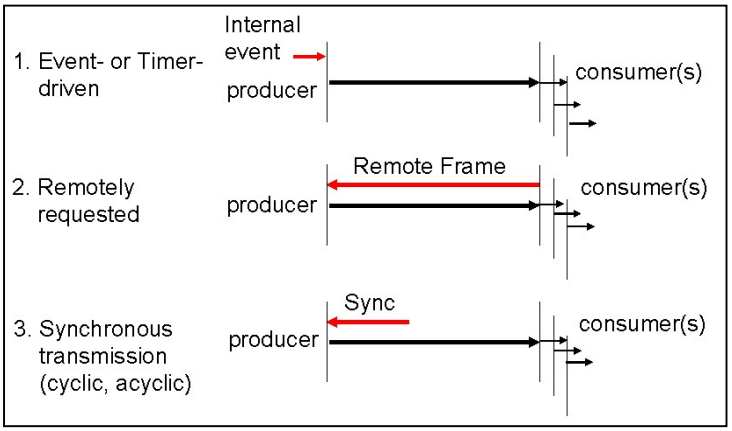


Fig 1.: PDO Transmission principles

**PDO Identifier**

The most important communication parameter in a PDO is the CAN identifier (also know as the communication object identifier, or COB-ID). It is used to identify the data, and determines their priority for bus access. For each CAN data telegram there may only be one sender node (producer), although all messages sent in the CAN broadcast procedure can be received by any number of nodes (consumers). Thus a node can make its input information available to a number of bus devices at the same time - even without transferring them through a logical bus master.

**Default Identifier**

For the first four PDOs (PDO1…PDO4) CANopen provides default identifiers depending on the node address, but all other PDOs must have identifiers assigned to them. In the system of default identifiers, all the nodes (here: slaves) communicate with one central station (the master), since slave nodes do not listen by default to the send identifier of other slave nodes: If the consumer-producer model of CANopen PDOs is to be used for direct data exchange between nodes (without a master), the distribution of identifiers must be appropriately adapted, so that the TxPDO identifier of the producer agrees with the RxPDO identifier of the consumer: This procedure is known as PDO linking. It permits, for example, easy construction of electronic drives in which several slave axes simultaneously listen to the actual value in the master axis TxPDO.

PDOs can be activated or deactivated by changing the most significant bit of the identifier parameter. Depending on the implementation, modifying the identifier may be limited to the pre-operational state of the device or even be not supported at all.

**PDO Transmission Types**

CANopen offers a number of possible ways to transmit process data (see fig. 1).

**Event driven:**

The “event” is the alteration of an input value, the data being transmitted immediately after this change. The event-driven flow can make optimal use of the bus bandwidth, since instead of the whole process image it is only the changes in it that are transmitted. A short reaction time is achieved at the same time, since when an input value changes it is not necessary to wait for the next interrogation from a master.

As from CANopen Version 4 it is possible to combine the event driven type of communication with a cyclic update. Even if an event has not just occurred, event driven TxPDOs are sent after the event timer has elapsed. If an event does occur, the event timer is reset. For RxPDOs the event timer is used as a watchdog in order to monitor the arrival of event driven PDOs. If a PDO does not arrive within a set period of time, the bus node adopts the error state.

**Polled**

The PDOs can also be polled by data request telegrams (remote frames). In this way it is possible to get the input process image of event-driven inputs onto the bus, even when the inputs have not changed,

for instance by a monitoring or diagnostic device brought into the network while it is running. The time needed to react on a remote frames and send the requested PDO depends on the CAN controller used: FullCAN Controllers with fully integrated frame filtering typically respond immediately to remote frames and send whatever is in the corresponding data buffer. Here the local application program (or firmware) has to make sure that the data is updated frequently, and one cannot tell how old the data is. CAN Controllers with simplified frame filtering (BasicCAN) typically forward the remote request to the application firmware, which then compiles the PDO with actual data. This takes longer, but ensures “fresh” data.

As this device behaviour normally is not transparent to the user and as there are CAN controllers that do not support remote frames at all, polling is not a preferable PDO communication method.

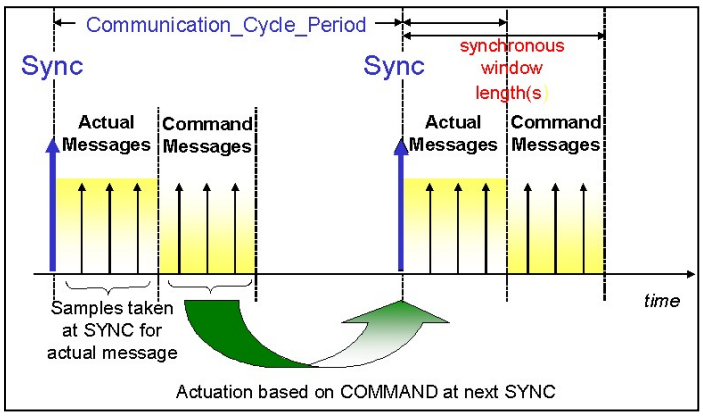


Fig 2.: CANopen Sync Mechanism

**Synchronised**

It is not only for drive applications that it is worth while to synchronise the sampling of the input information and the setting the outputs. For this purpose CANopen provides the SYNC object, a CAN telegram of high priority but containing no user data, whose reception is used by the synchronised nodes as a trigger for reading the inputs or for setting the outputs (fig. 2).

The “PDO transmission type” parameter specifies how the transmission of the PDO is triggered, or how received PDOs are handled:

**Acyclic Synchronous**

PDOs of transmission type 0 function synchronously, but not cyclically. An RxPDO is only evaluated after the next SYNC telegram has been received. In this way, for instance, axis groups can be given new target positions one after another, but these positions only become valid at the next SYNC - without the need to be constantly outputting reference points. A device whose TxPDO is configured for transmission type 0 acquires its input data when it receives the SYNC (synchronous process image) and then transmits it if the data correspond to an event (such as a change in input) having occurred. Transmission type 0 thus combines transmission for reasons that are event driven with a time for transmission (and, as far as possible, sampling) and processing given by the reception of "SYNC".

**Cyclic Synchronous**

In transmission types 1-240 the PDO is transmitted cyclically: after every ”n-th” SYNC (n = 1...240). Since transmission types can be combined on a device as well as in the network, it is possible, for example, for a fast cycle to be agreed for digital inputs (n = 1), whereas the data for analog inputs is transmitted in a slower cycle (e.g. n = 10). RxPDOs do not generally distinguish between transmission types 0...240: a PDO that has been received is set to valid when the next SYNC is received. The cycle time (SYNC rate) can be monitored (object 0x1006), so that if the SYNC fails the device reacts in accordance with the definition in the device profile, and switches, for example, its outputs into the fault state.

### Inhibit time

When selecting the type of event-driven PDO communication, consideration must be given to the fact that in certain circumstances many events occur at the same time, resulting in corresponding delays before a relatively low priority PDO can be transmitted. The possibility of a continuously changing input with a high PDO priority monopolising the bus (the "babbling idiot") must also be prevented. For this reason, event-driven mode is switched off by default for analogue inputs, in accordance with the CANopen specification, and has to be activated by means of object 0x6423. The "inhibit time" parameter (fig. 3) can be used to implement a "transmit filter" that does not increase the reaction time for relatively new input alterations, but is active for changes that follow immediately afterwards.

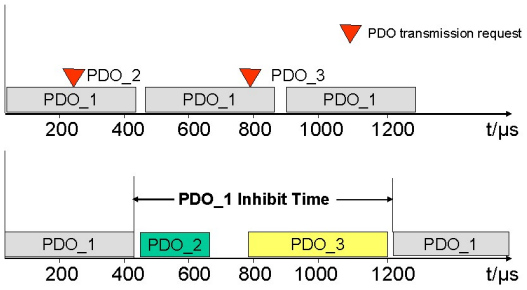


Fig 3.: Inhibit Time

The inhibit time (transmit delay time) specifies the minimum length of time that must be allowed to elapse between the transmission of two of the same telegrams. If the inhibit time is used, the maximum bus loading can be determined, so that the worst case latency can then be found.

### PDO Parameterisation

Even though the majority of CANopen networks operate satisfactorily with the default settings, i.e. with the minimum of configuration effort, it is wise at least to check whether the existing bus loading is reasonable: 70-80% bus loading may be acceptable for a network operating purely in cyclic synchronous modes, but for a network with event-driven traffic this value would generally be too high, as there is hardly any bandwidth available for additional events.

### Application Requirements

The communication of the process data must be optimised in the light of application requirements which are likely to be to some extent in conflict. These include

- Little work on parameterisation - useable default values are optimal
- Guaranteed reaction time for specific events
- Cycle time for control loops closed over the bus
- Safety margins for bus malfunctions (enough bandwidth for the repetition of messages)
- Maximum baud rate - depends on the maximum bus length
- Desired communication paths - who is speaking with whom

In order to determine the requirements of the application the cycle time of the control task(s) should be considered. For each device or signal type it should be defined if the reaction time or the cycle time is crucial.

Typical is for

- Controlled axis: cycle time and synchronicity
- Positioned axis: reaction time, simultaneous start
- Digital I/O: reaction time
- Analogue I/O: cycle time

Furthermore, the switch-off time in case of communication failure has to be considered: how long may it take to detect such a situation? This time determines the cycle time (and life time factor) of the node supervision (Guarding/Heartbeat).

A important factor for the layout of the network often turns out to be the available bus bandwidth (bus load).

**Select the Baudrate**

We generally begin by choosing the highest baud rate that the bus will permit. It should be borne in mind that serial bus systems are always more sensitive to interference at higher baud rates, so the better rule is "just as fast as needed". 1000 kbit/s are not usually necessary, and only to be unreservedly recommended on networks within a control cabinet where there is no electrical isolation between the bus nodes. Experience also tends to show that estimates of the length of bus cable laid are often over-optimistic - the length actually laid tends to be longer.

**Determine the Communication Type**

Once the baud rate has been chosen it is appropriate to specify the PDO communication type(s). These have different advantages and disadvantages:

Cyclic synchronous communication provides an accurately predictable bus loading, and therefore a defined time behaviour - you could say that the standard case is the worst case. It is easy to configure: The SYNC rate parameter sets the bus loading globally. The process images are synchronised: Inputs are read at the same time, output data is set valid simultaneously, although the quality of the synchronisation depends on the implementation.

Only few CANopen master cards are capable of synchronising the CANopen bus system with the cycles of the application program (PLC or NC) – the Beckhoff cards provide this feature. The guaranteed reaction time under cyclic synchronous communication is always at least as long as the cycle time, and the bus bandwidth is not exploited optimally, since "old" data, i.e. data that has not changed, is continuously transmitted. It is however possible to optimise the network through the selection of different SYNC multiples (transmission types 1...240), so that data that changes slowly is transmitted less often than, for instance, time-critical inputs.

It must, however, be borne in mind that input states that last for a time that is shorter than the cycle time will not necessarily be communicated. If it is necessary for such conditions to be registered, the associated PDOs for asynchronous communication should be provided.

Event-driven asynchronous communication is optimal from the point of view of reaction time and the exploitation of bus bandwidth - it can be described as "pure CAN". Your choice must, however, also take account of the fact that it is not impossible for a large number of events to occur simultaneously, leading to corresponding delays before a PDO with a relatively low priority can be sent. Proper network planning therefore necessitates a worst-case analysis.

Through the use of, for instance inhibit time it is also necessary to prevent a constantly changing input with a high PDO priority from blocking the bus. As mentioned before, this is the reason that event driving is switched off by default for analogue inputs, and must be turned on specifically. Time windows for the transmit PDOs can be set using event timers: the telegram is not sent again before the inhibit time has elapsed, and not later than the time required for the event timer to complete (see fig. 4)

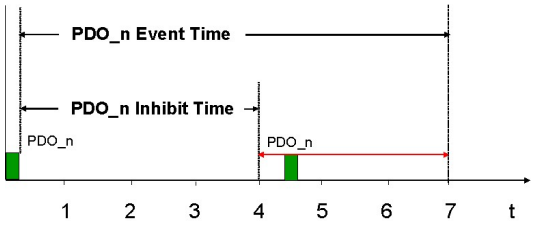


Fig 4: TxPDO Time Window

It is also possible to combine the two main PDO principles. It can, for instance, be helpful to exchange the command and actual values of an axis controller synchronously, while limit switches, or motor temperatures with limit values are monitored with event-driven PDOs. This combines the advantages of the two principles: synchronicity for the axis communication and short reaction times for limit switches. In spite of being event-driven, the distributed limit value monitoring avoids a constant addition to

the bus load from the analogue temperature value. In this example it can also be of value to deliberately manipulate the identifier allocation, in order to optimise bus access by means of priority allocation: the highest priority is given to the PDO with the limit switch data, and the lowest to that with the temperature values. Optimisation of bus access latency time through modification of the identifier allocation is not, however, normally required. On the other hand the identifiers must be altered if "masterless" communication is to be made possible (PDO linking). In this example it would be possible for one RxPDO for each axis to be allocated the same identifier as the limit switch TxPDO, so that alterations of this important input value can be received without delay.

### Determining the Bus Load

It is always worth determining the bus load. But what bus load values are "permitted", or indeed sensible? It is first necessary to distinguish a short burst of telegrams in which a number of CAN messages follow one another immediately - a temporary 100% bus loading. This is only a problem if the sequence of receive interrupts that it caused at the CAN nodes can not be handled. This would constitute a data overflow (or "CAN queue overrun"). This can occur at very high baud rates (> 500 kbit/s) at nodes with software telegram filtering and relatively slow or heavily loaded microcontrollers if, for instance, a series of remote frames (which do not contain data bytes, and are therefore very short) follow each other closely on the bus (at 1 Mbit/s this can generate an interrupt every 40 µs; for example, an NMT master might transmit all its guarding requests in an unbroken sequence). This can be avoided through skilled implementation, and the user should be able to assume that the device suppliers have taken the necessary precautions. A burst condition is entirely normal immediately after the SYNC telegram, for instance: triggered by the SYNC, all the nodes that are operating synchronously try to send their data at almost the same time. A large number of arbitration processes take place, and the telegrams are sorted in order of priority for transmission on the bus. This is not usually critical, since these telegrams do contain some data bytes, and the telegrams trigger a sequence of receive interrupts at the CAN nodes which is indeed rapid, but is nevertheless manageable.

The Bus load most often refers to the value averaged over several primary cycles, that is the mean value over 100-500 ms. CAN, and therefore CANopen, is indeed capable of managing a bus loading of close to 100% over long periods, but this implies that no bandwidth is available for any repetitions that may be necessitated by interference, for asynchronous error messages, parameterisation and so on. Clearly, the dominant type of communication will have a large influence on the appropriate level of bus loading: a network with entirely cyclic synchronous operation is always in any case near to the "worst case" state, and can therefore be operated with values in the 70-80% range. The figure is very hard to state for an entirely event-driven network: an estimate must be made of how many events additional to the current state of the system might occur, and of how long the resulting burst might last - in other words, for how long the lowest priority message will be delayed. If this value is acceptable to the application, then the current bus loading is acceptable. As a rule of thumb it can usually be assumed that an event-driven network running with a base loading of 30-40% has enough reserve for worst-case scenarios, but this assumption does not obviate the need for a careful analysis if delays could have critical results for the plant. The Beckhoff FC510x PC cards indicate the bus loading via the System Manager (see Fig. 5).

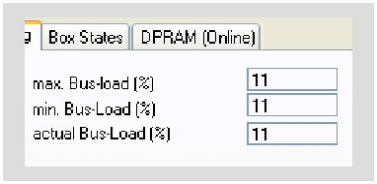


Fig 5.: Bus load Indication

This variable can also be processed in the PLC, or can be displayed in the visualisation system.

These considerations lead to the following configuration guideline and procedure for the configuration of a CANopen network:

1. Start with the following default settings:
- digital I/Os: event driven
- analogue I/Os: cyclic (synchronous)
- Servo drives: cyclic synchronous
- Variable speed drives: command valued acyclic synchronous, actual values cyclic synchronous
- Bus cycle time (sync rate) = task cycle time

2. Now make a check on the bus load

In more than 80% of the applications the configuration is now finished

3. If the bus load is very low (< 30%):
- Reduce sync rate (= bus cycle time)
- Consider to communicate digital I/O synchronously, if cycle time is shorter than minimal state time of inputs
- Possibly reduce baud rate

4. If the bus load is very high (>70%)
- Check sync rate (bus cycle time) – possibly it can be increased
- Consider increasing sync multiple of analogue I/O
- If possible: increase baud rate
- Maybe it is necessary to introduce a second CANopen network

In any case it is important to measure and check the bus load!

Martin Rostan
Beckhoff
Ostendstr. 196, D- 90482 Nürnberg
+49 911 54056-11
+49 911 54056-29
m.rostan@beckhoff.com
www.beckhoff.com