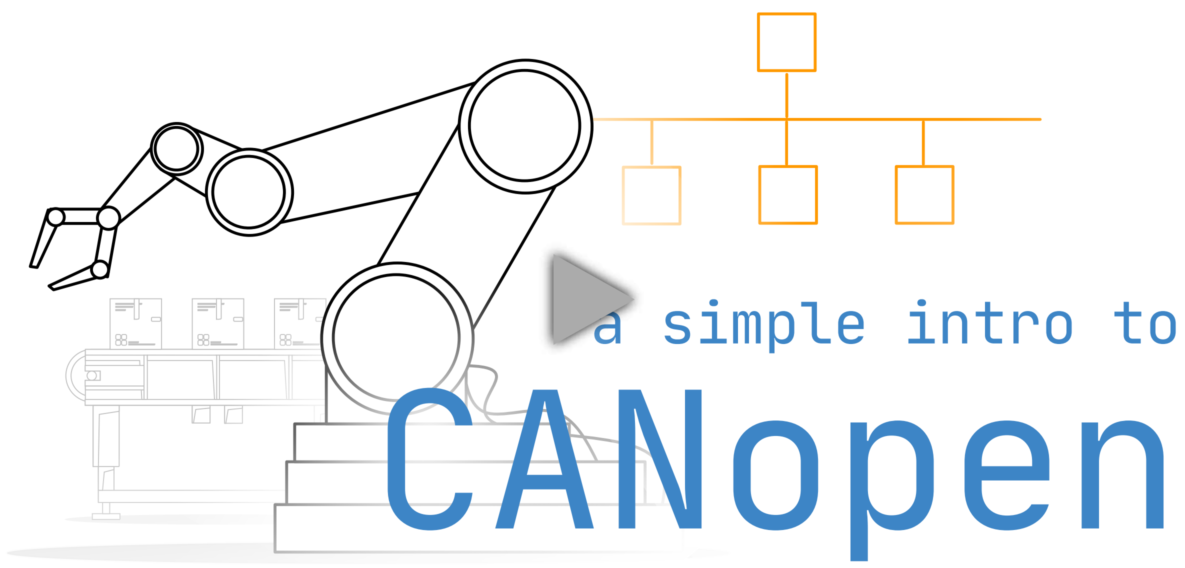
CANopen Explained - A Simple Intro (2020)



Need a simple, practical intro to CANopen?

In this guide we introduce the CANopen protocol basics incl. the object dictionary, services, SDO, PDO and master/slave nodes.

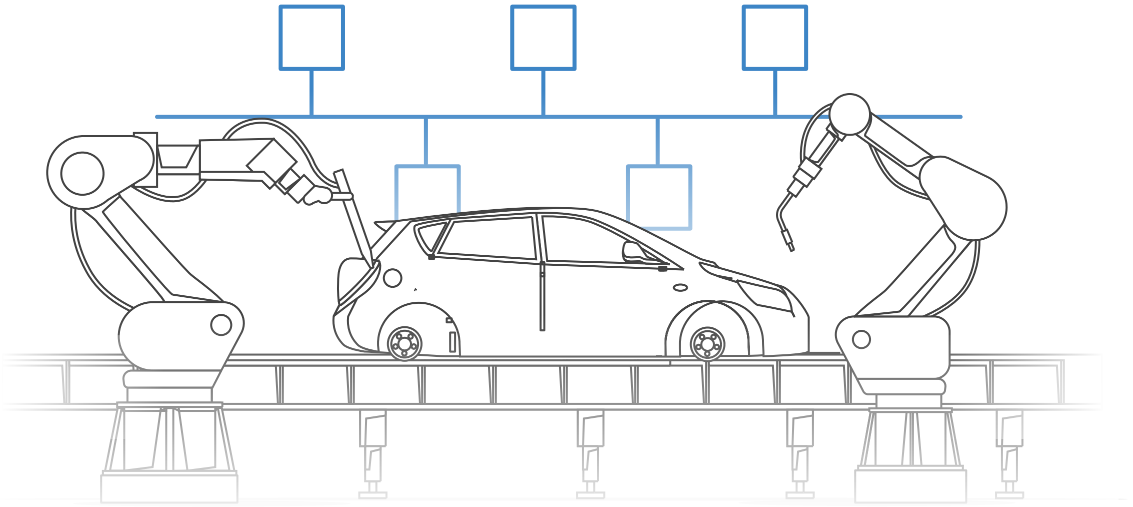
Note: **CANopen can seem complex** - so this tutorial is a visual intro in layman's terms.

Read on below to fully understand CANopen.

*See also our **20 min CANopen intro video** above.*

In this article

1. What is CANopen?
2. Six core CANopen concepts
3. CANopen communication basics
4. The Object Dictionary
5. SDOs: Service Data Objects
6. PDOs: Process Data Objects
7. CANopen data logging use cases



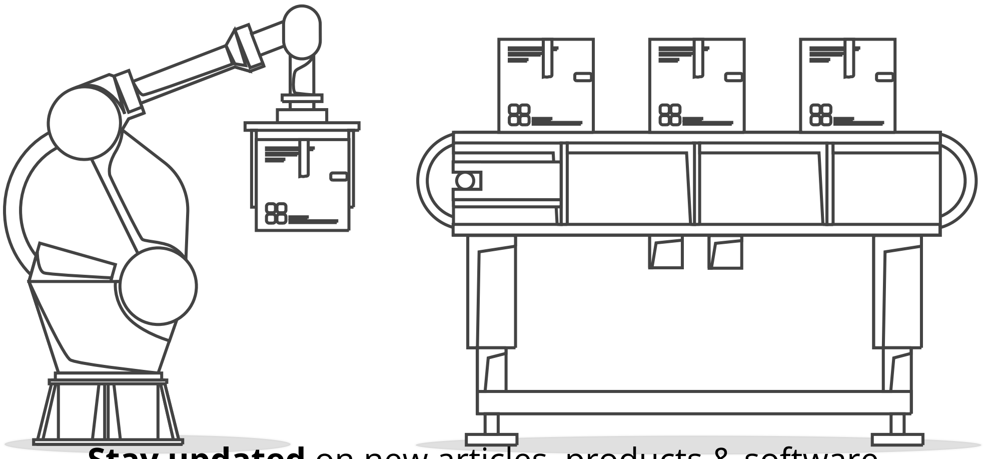
What is CANopen?

CANopen is a CAN based communication protocol.

The CANopen standard is useful as it enables off-the-shelf interoperability between devices (nodes) in e.g. industrial machinery. Further, it provides standard methods for configuring devices - also after installation.

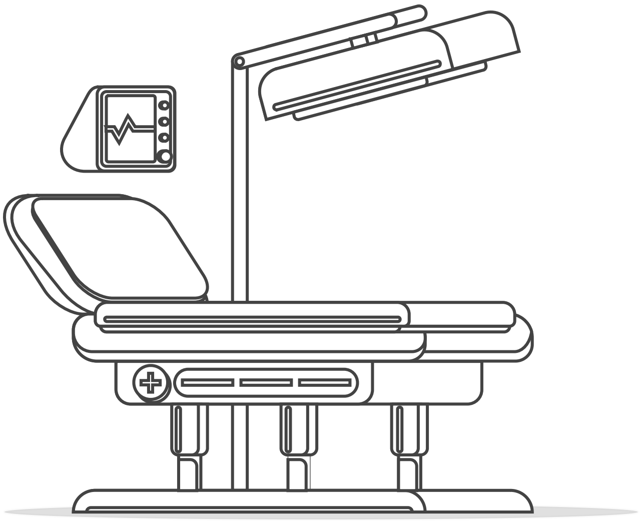
CANopen was originally designed for motion-oriented machine control systems.

Today, CANopen is extensively used in motor control (stepper/servomotors) - but also a wide range of other applications:



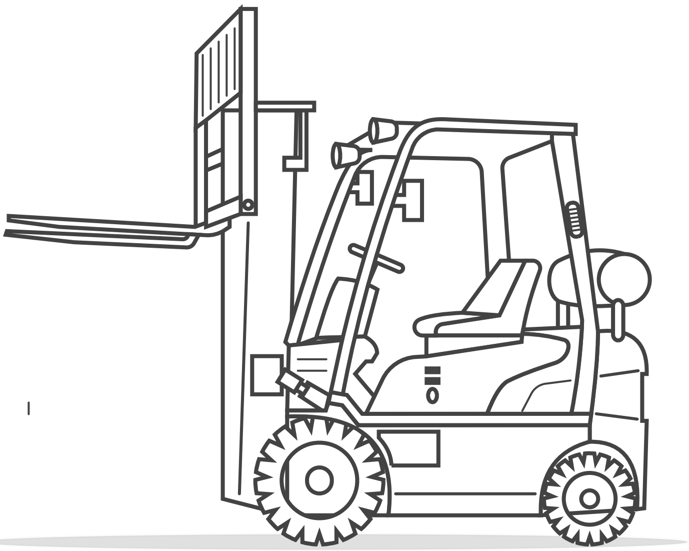
Robotics

Automated robotics, conveyor belts & other industrial machinery



Medical

X-ray generators, injectors, patient tables & dialysis devices



Automotive

Agriculture, railway, trailers, heavy duty, mining, marine & more

CANopen - higher layer protocol

The following is important to understand:

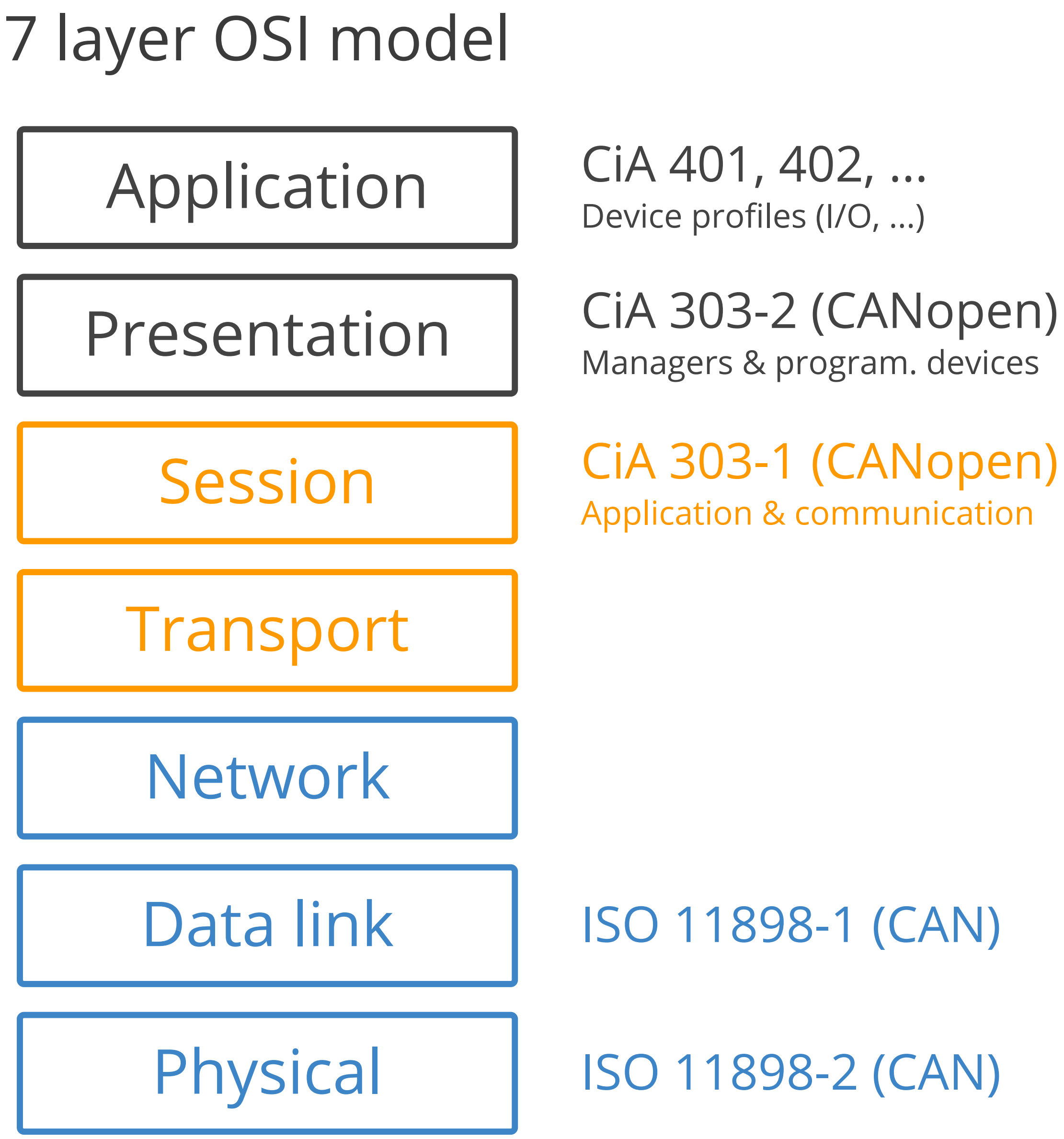
CANopen is a "higher layer protocol" based on CAN bus.

This means that CAN bus (ISO 11898) serves as the 'transport vehicle' (like a truck) for CANopen messages (like containers).

You can view CANopen from a 7-layer OSI model, see below.

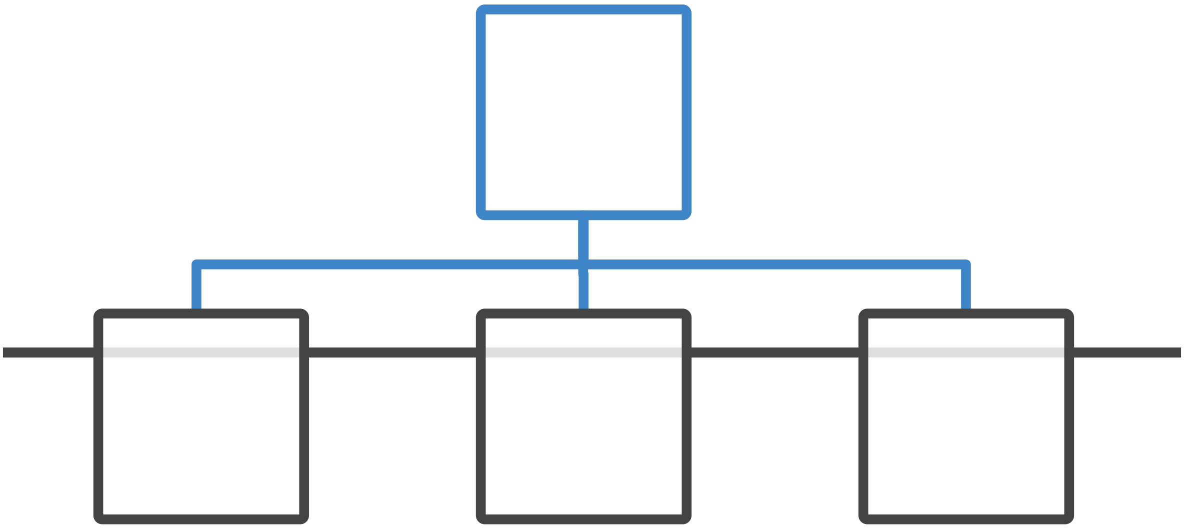
CANopen in OSI model context +

CANopen FD +



Six core CANopen concepts

Even if you're familiar with CAN bus and e.g. J1939, CANopen adds a range of important new concepts:



Communication Models

There are 3 models for device/node communication: Master/slave, client/server and producer/consumer

Communication Protocols

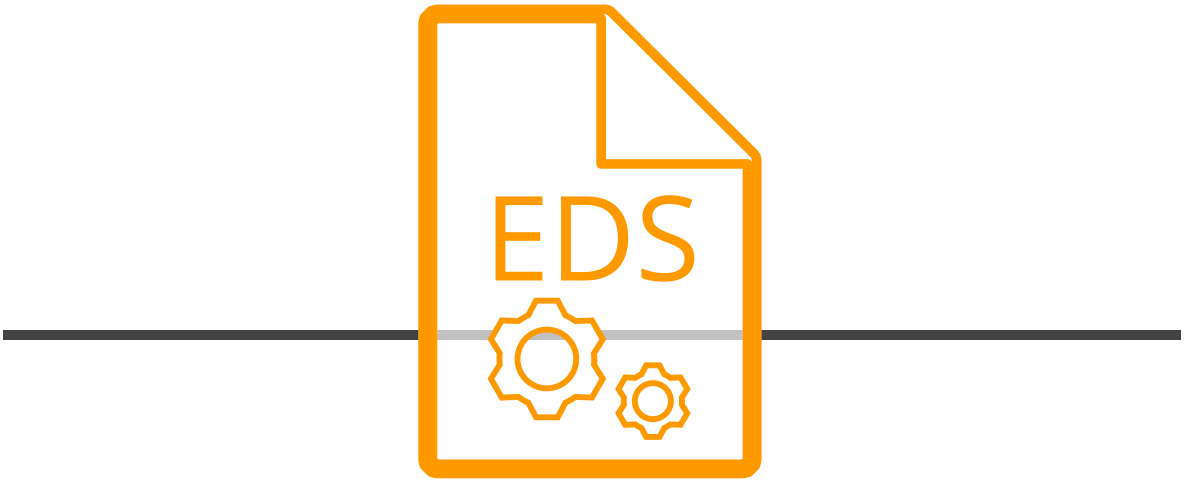
Protocols are used for communication, e.g. configuring nodes (SDOs) or transmitting real-time data (PDOs)

Device States

A device supports different states. A 'master' node can change state of a 'slave' node - e.g. resetting it

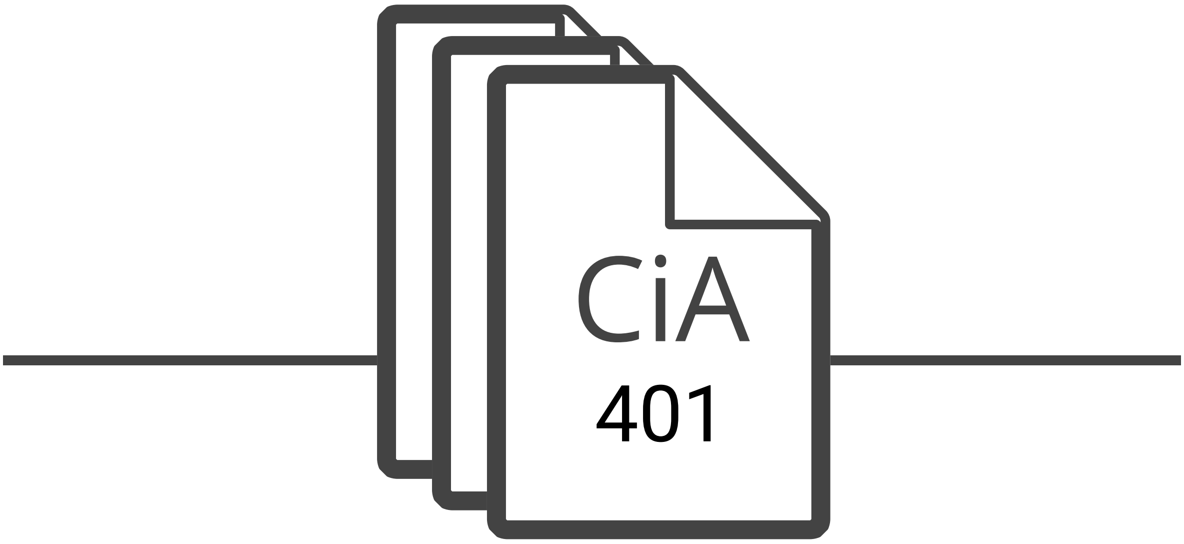
Object Dictionary

Each device has an OD with entries that specify e.g. the device config. It can be accessed via SDOs



Electronic Data Sheet

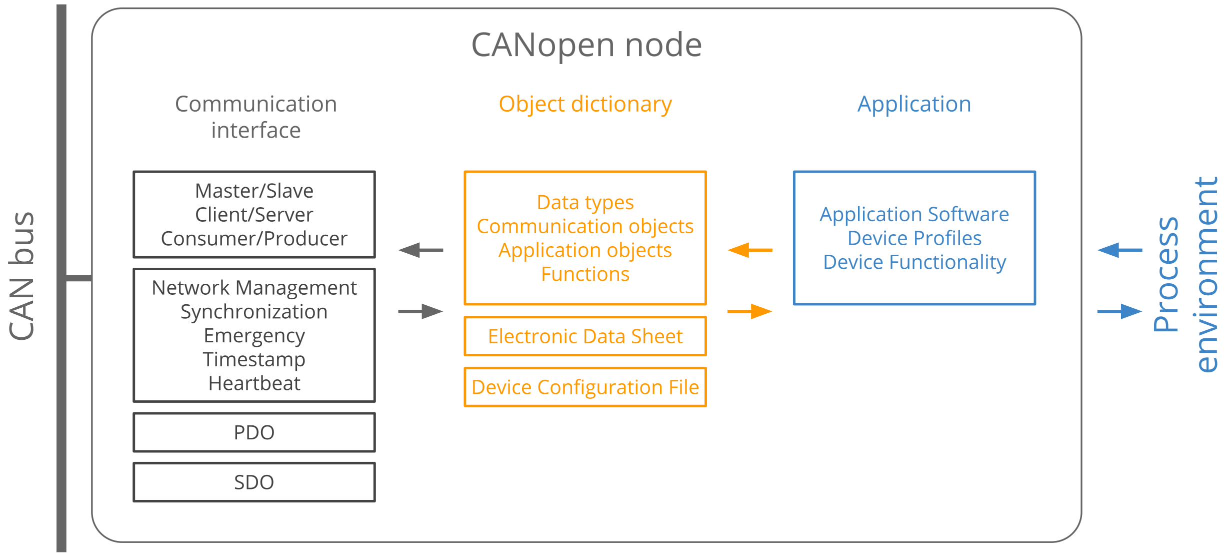
The EDS is a standard file format for OD entries - allowing e.g. service tools to update devices



Device Profiles

Standards describe e.g. I/O modules (CiA 401) and motion-control (CiA 402) for vendor independence

The below illustration shows how the CANopen concepts link together - and we will detail each below:

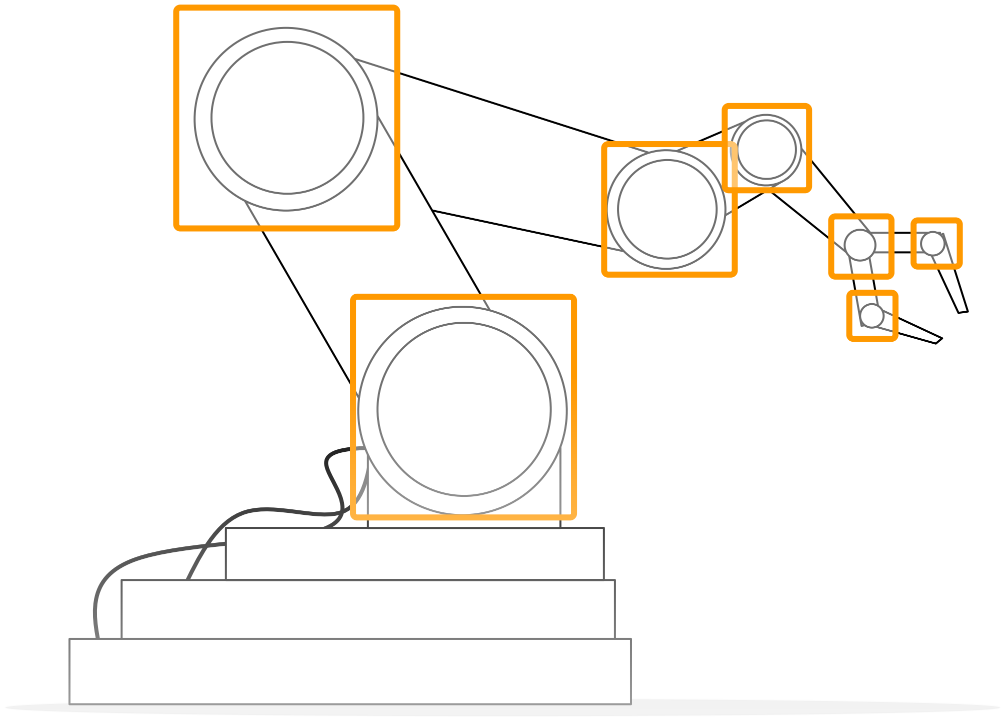


CANopen communication basics

In a CANopen network, several devices need to communicate.

For example, in an industrial automation setup you may have a robot arm with multiple servomotor nodes and a control interface/PC node.

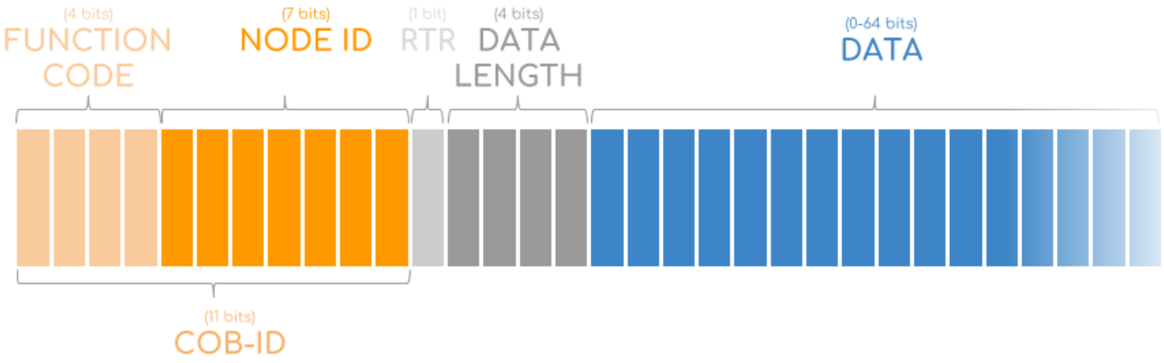
To facilitate communication, **three models exist** within CANopen - each closely linked to the CANopen protocols that we look at shortly. See below for a brief introduction:



CANopen communication models +

The CANopen frame

To understand CANopen communication, it is necessary to break down the **CANopen CAN frame**:



The 11-bit CAN ID is referred to as the Communication Object Identifier (**COB-ID**) and is split in two parts:
By default, the first 4 bits equal a **function code** and the next 7 bits contain the **node ID**.

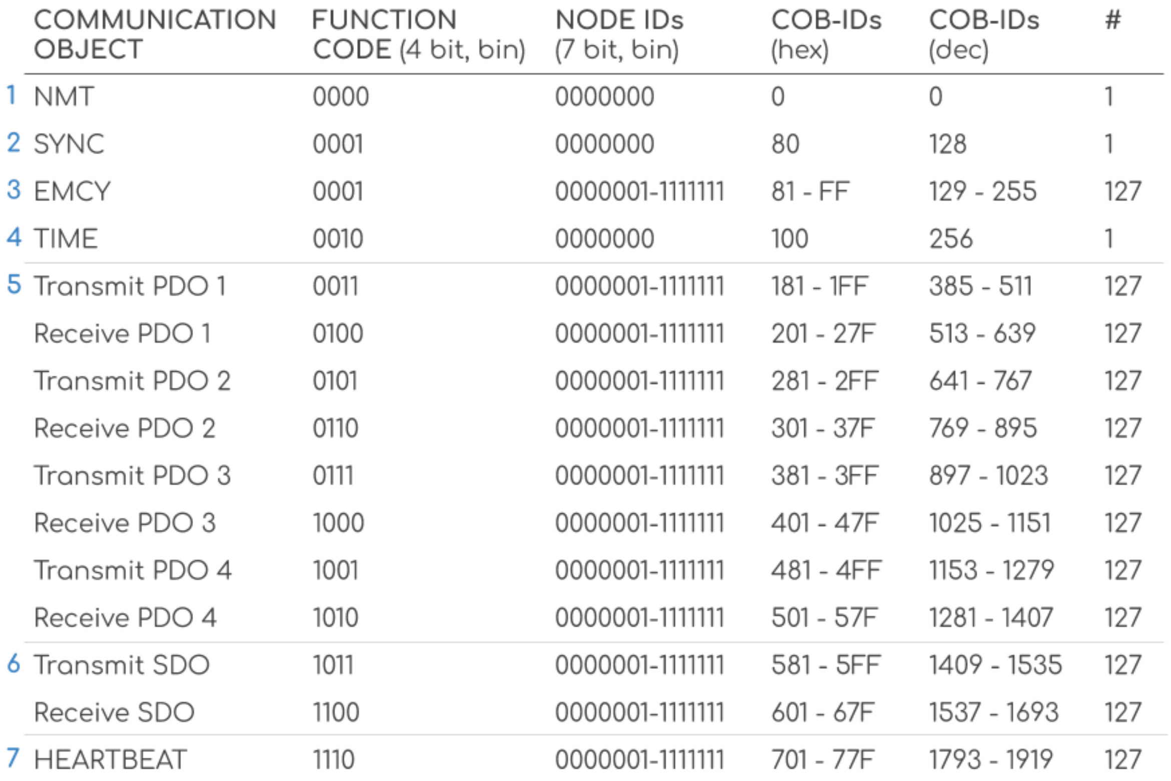
To understand how the COB-ID works, let's take outset in the pre-defined allocation of identifiers used in simple CANopen networks (see the table).

Note: We'll refer to COB-IDs and Node IDs in HEX below.

As evident, the COB-IDs (e.g. 381, 581, ...) are linked to the communication services (transmit PDO 3, transmit SDO, ...).

As such, **the COB-ID details which node is sending/receiving data - and what service is used**.

Example +



	COMMUNICATION OBJECT	FUNCTION CODE (4 bit, bin)	NODE IDs (7 bit, bin)	COB-IDs (hex)	COB-IDs (dec)	#
1	NMT	0000	0000000	0	0	1
2	SYNC	0001	0000000	80	128	1
3	EMCY	0001	0000001-1111111	81 - FF	129 - 255	127
4	TIME	0010	0000000	100	256	1
5	Transmit PDO 1	0011	0000001-1111111	181 - 1FF	385 - 511	127
	Receive PDO 1	0100	0000001-1111111	201 - 27F	513 - 639	127
	Transmit PDO 2	0101	0000001-1111111	281 - 2FF	641 - 767	127
	Receive PDO 2	0110	0000001-1111111	301 - 37F	769 - 895	127
	Transmit PDO 3	0111	0000001-1111111	381 - 3FF	897 - 1023	127
	Receive PDO 3	1000	0000001-1111111	401 - 47F	1025 - 1151	127
	Transmit PDO 4	1001	0000001-1111111	481 - 4FF	1153 - 1279	127
	Receive PDO 4	1010	0000001-1111111	501 - 57F	1281 - 1407	127
6	Transmit SDO	1011	0000001-1111111	581 - 5FF	1409 - 1535	127
	Receive SDO	1100	0000001-1111111	601 - 67F	1537 - 1693	127
7	HEARTBEAT	1110	0000001-1111111	701 - 77F	1793 - 1919	127

CANopen COB-ID converter

Enter hex ID (e.g. 26A)

Online CANopen COB-ID converter

Our COB-ID converter lets you quickly look up a CANopen COB-ID to return basic details incl. function code and node ID.

CANopen communication protocols/services

Below we briefly outline the 7 service types mentioned, incl. how they utilize the 8 *CAN frame data bytes*.

#1 Network Management (NMT) +

#2 Synchronization (SYNC) +

#3 Emergency (EMCY) +

#4 Timestamp (TIME) [PDO] +

#5 Process Data Object [PDO] +

#6 Service Data Object [SDO] +

#7 Node monitoring (Heartbeat) [SDO] +

The PDO and SDO services are particularly important as they form the basis for most CANopen communication.

Below we deep-dive on each of these - but first we need to introduce a core concept of CANopen: The **object dictionary**.

CANopen Object Dictionary

All CANopen nodes must have an object dictionary (OD) - but what is it?

*The **object dictionary** is a standardized structure containing all parameters describing the behavior of a CANopen node.*

OD entries are looked up via a 16-bit index and 8-bit subindex. For example, index 1008 (subindex 0) of a CANopen-compliant node OD contains the node *device name*.

Specifically, an entry in the object dictionary is defined by attributes:

- **Index:** 16-bit base address of the object
- **Object name:** Manufacturer device name
- **Object code:** Array, variable, or record
- **Data type:** E.g. VISIBLE_STRING, or UNSIGNED32 or Record Name
- **Access:** rw (read/write), ro (read-only), wo (write-only)
- **Category:** Indicates if this parameter is mandatory/optional (M/O)

OD INDEX (16 bits, hex)	DESCRIPTION
0000	Reserved
0001 - 025F	Data types
0260 - 0FFF	Reserved
1000 - 1FFF	Communication object area
2000 - 5FFF	Manufacturer specific area
6000 - 9FFF	Device profile specific area
A000 - BFFF	Interface profile specific area
C000 - FFFF	Reserved

OD standardized sections

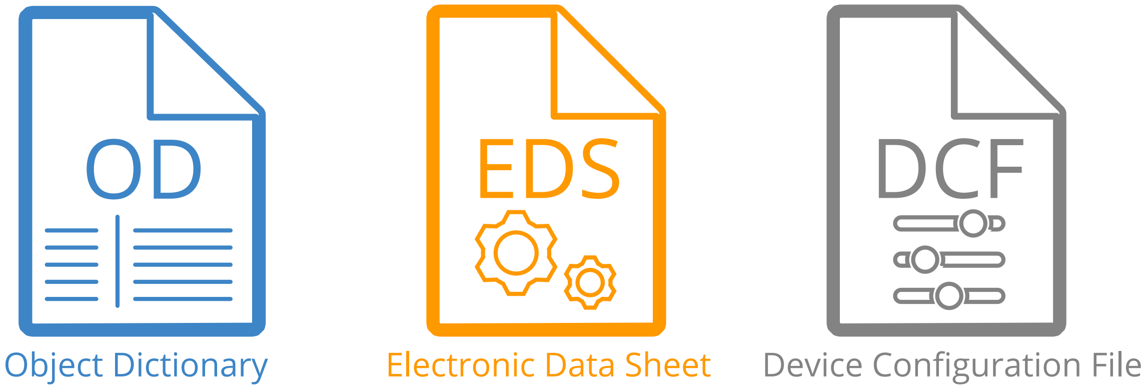
The object dictionary is split into **standardized sections** where some entries are mandatory and others are fully customizable.

Importantly, OD entries of a device (e.g. a slave) can be accessed by another device (e.g. a master) via CAN using e.g. SDOs.

For example, this might let an application master change whether a slave node logs data via a specific input sensor - or how often the slave sends a heartbeat.

Link to Electronic Data Sheet and Device Configuration File

To understand the OD, it is helpful to look at the 'human-readable form': The Electronic Data Sheet and Device Configuration File.



The Electronic Data Sheet (EDS)

In practice, configuring/managing complex CANopen networks will be done using adequate software tools.

To simplify this, the CiA 306 standard defines a human-readable (and machine friendly) INI file format, acting as a "template" for the OD of a device - e.g. the "ServoMotor3000". This EDS is typically provided by the vendor and contains *info* on all device objects (but not values).

Device Configuration File (DCF)

Assume a factory has bought a ServoMotor3000 to integrate into their conveyor belt. In doing so, the operator edits the device EDS and adds *specific parameter values* and/or changes the names of each object described in the EDS.

In doing so, the operator effectively creates what is known as a Device Configuration File (DCF). With this in place, the ServoMotor3000 is ready for integration into the specific CANopen network on-site.

EDS and DCF example +

As mentioned, the DCF is typically created upon device integration. However, often it will be necessary to read and/or change the object values of a node **after initial configuration** - this is where the CANopen SDO service comes into play.

SDO - configuring the CANopen network

What is the SDO service?

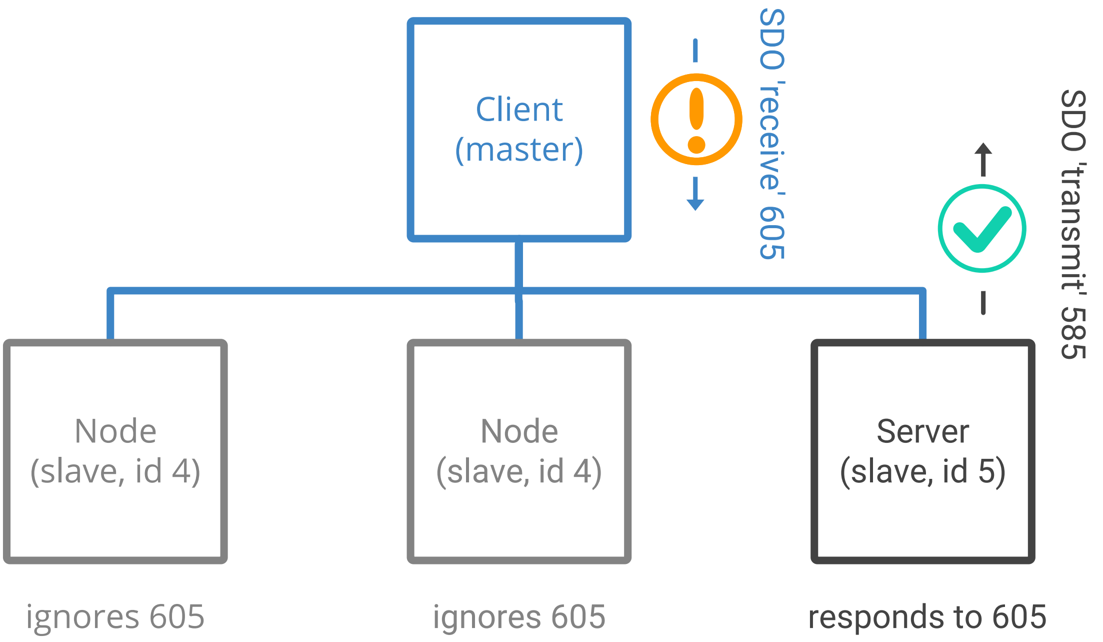
*The **SDO service** allows a CANopen node to read/edit values of another node's object dictionary over the CAN network.*

As mentioned under 'communication models', the CANopen SDO services utilize a "client/server" behavior.

Specifically, an SDO "client" initiates the communication with one dedicated SDO "server".

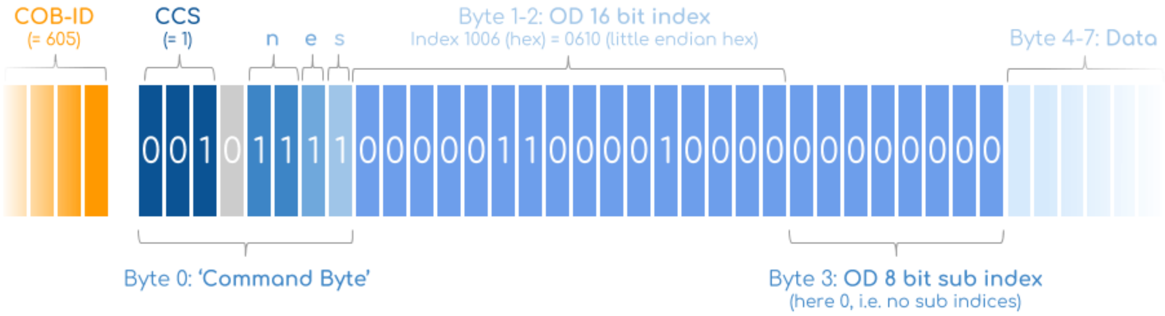
The purpose can be to update an OD entry (called an "SDO download") or read an entry ("SDO upload").

In simple master/slave networks, the node with NMT master functionality acts as the client for all NMT slave nodes reading or writing to their ODs.



Example: Client node SDO download

The client node can initiate an SDO download to node 5 by broadcasting below CAN frame - which will trigger node 5 (and be ignored by other nodes, see above illustration). The SDO 'receive' (i.e. request) CAN frame looks as below:



SDO message variables explained +

CANopen SDO example comments +

SDOs are flexible, but carry a lot of overhead - making them less ideal for real-time operational data. This is where the PDO comes in.

PDO - operating the CANopen network

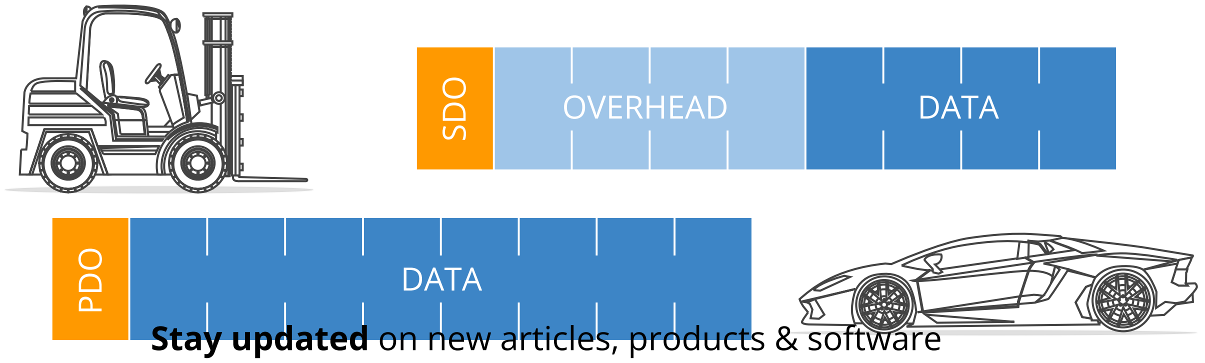
First of all: What is the CANopen PDO service?

*The CANopen **PDO service** is used for effectively sharing real-time operational data across CANopen nodes.*

For example, the PDO would carry pressure data from a pressure transducer - or temperature data from a temperature sensor.

But wait: Can't the SDO service just do this?

Yes, in principle the SDO service could be used for this. However, a single SDO response can only carry 4 data bytes due to overhead (command byte and OD addresses). Further, let's say a master node needs two parameter values (e.g. "SensTemp2" and "Torque5") from Node 5 - to get this via SDO, it would require *4 full CAN frames* (2 requests, 2 responses).



In contrast, a PDO message can contain 8 full bytes of data - and it can contain multiple object parameter values within a single frame. Thus, what would require at least 4 frames with SDO could potentially be done with 1 frame in the PDO service.

The PDO is often seen as the most important CANopen protocol as it carries the bulk of information.

How does the CANopen PDO service work? +

PDO service vs. J1939 PGNs and SPNs +

CANopen data logging - use case examples

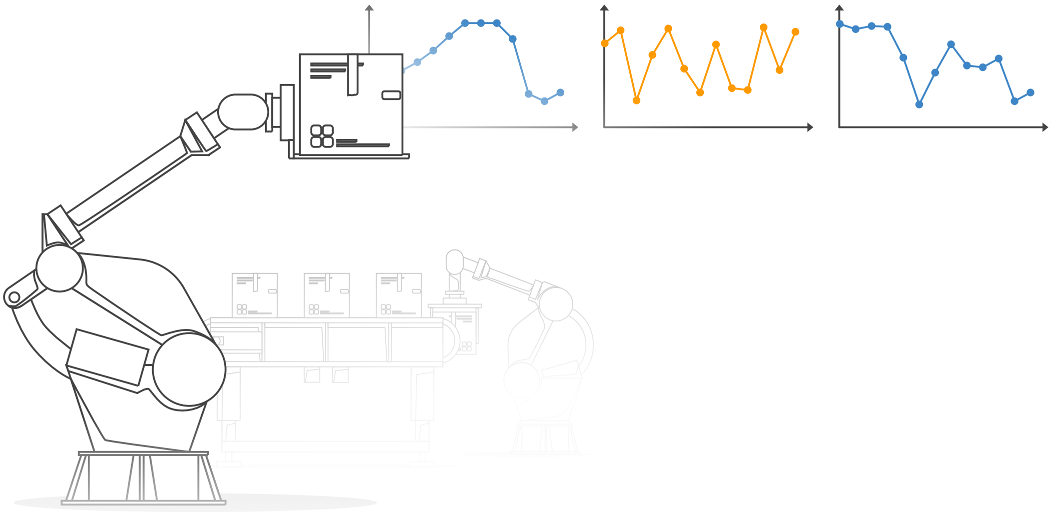
Since CANopen is a CAN-based protocol, it is possible to record raw CANopen frames using a CAN bus data logger.

As an example, the CANedge lets you record CANopen data to an 8-32 GB SD card. Simply connect it to your application to start logging - and process the data via free software/APIs.

learn more

Raw data decoding via CANopen DBC files +

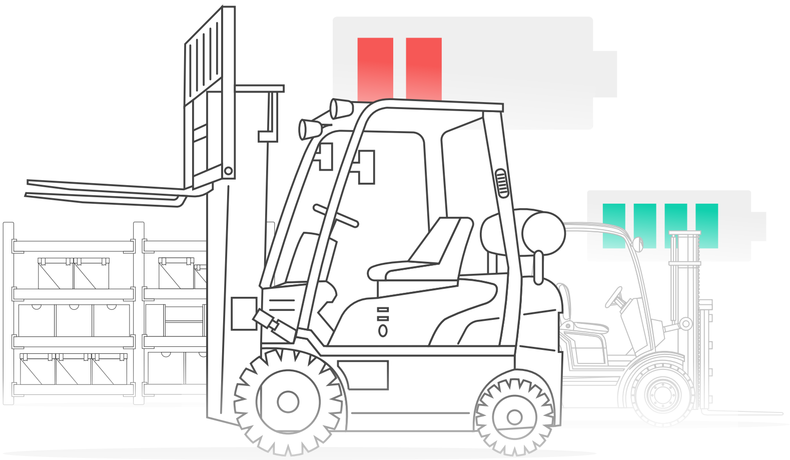
Solutions like the CANedge enable several CANopen logging use cases:



Logging CANopen node data

Generally, logging CANopen data can be used to e.g. analyze operational data. WiFi CAN loggers can also be used for e.g. over-the-air SDOs

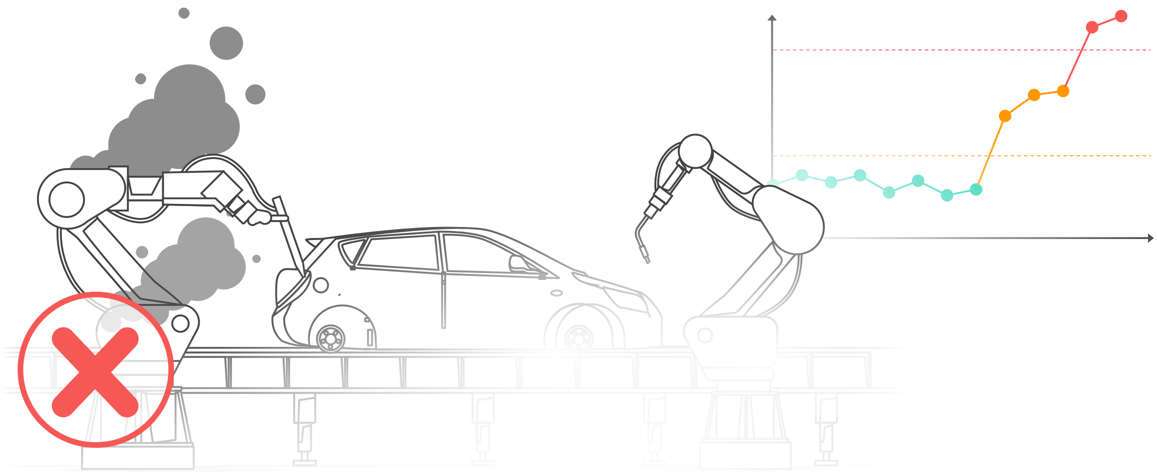
learn more →



Warehouse fleet management

CANopen is often used in EV forklifts/AGVs in warehouses, where monitoring e.g. SoC helps reduce breakdowns and improve battery life

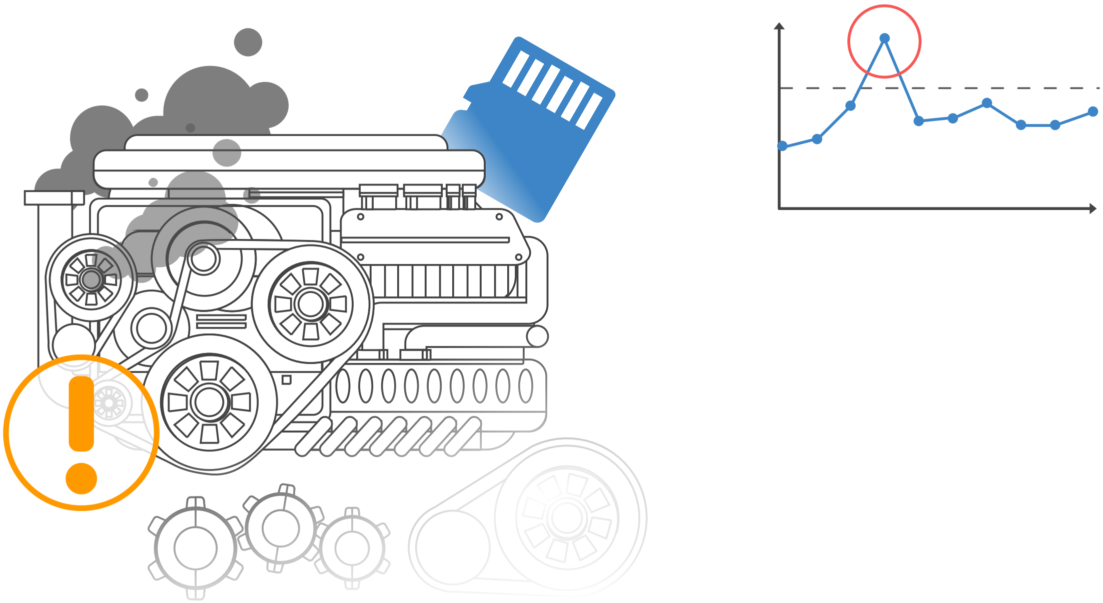
learn more →



Predictive maintenance

Industrial machinery can be monitored via IIoT CAN loggers in the cloud to predict and avoid breakdowns based on the CANopen data

learn more →



Machinery diagnostic blackbox

A CAN logger can serve as a 'blackbox' for industrial machinery, providing data for e.g. disputes or rare issue diagnostics

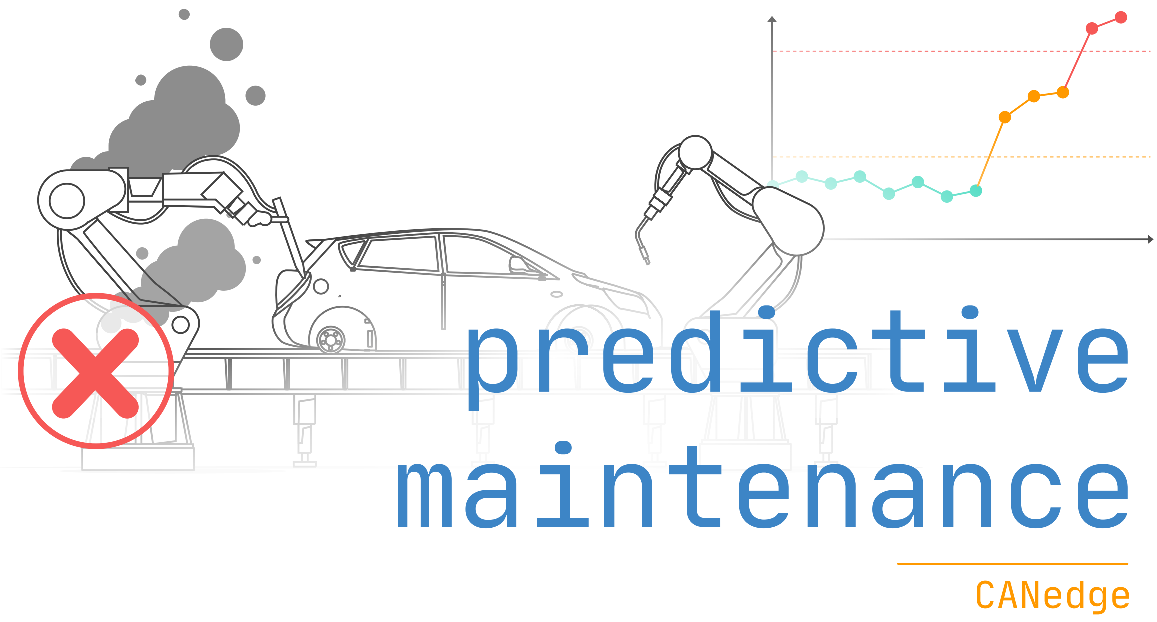
learn more →

Need to log/stream CANopen data?

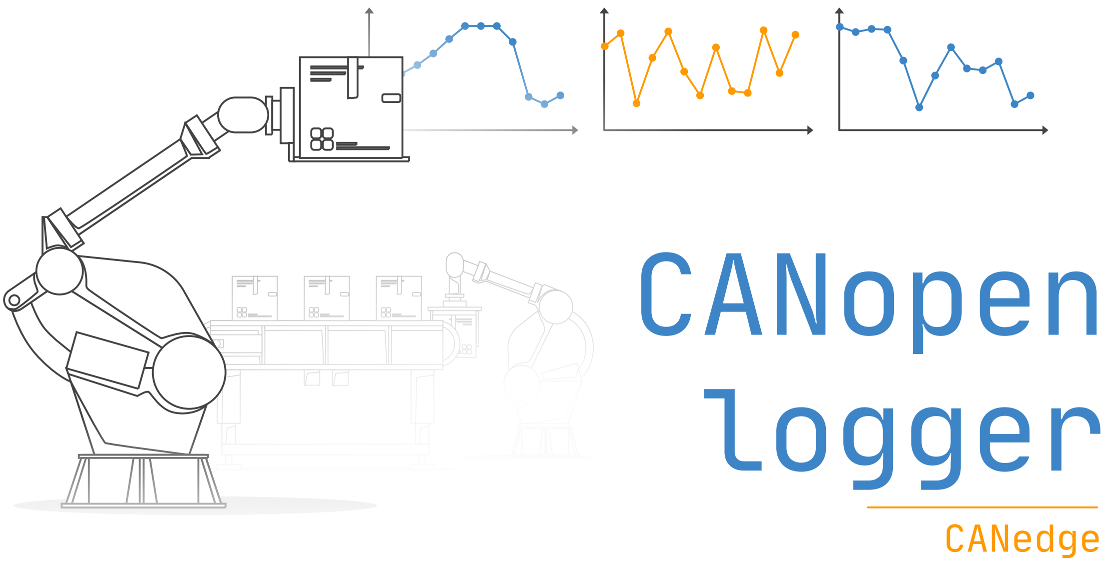
Get your CAN logger today!

buy now   contact us →

Recommended for you



PREDICTIVE MAINTENANCE 4.0



CANOPEN DATA LOGGER

CANEDGE2 - PRO CAN FD IoT LOGGER

Contact

CSS Electronics | VAT ID: DK36711949
Soeren Frichs Vej 38K (Office 35)
8230 Aabyhoej, Denmark

contact@csselectronics.com
+45 91 25 25 63

- Terms of Service
- Returns and Refunds
- Privacy Policy
- About Us
- CO2 Neutral

Newsletter

Email address

subscribe

Stay updated on new articles, products & software

✖