

CAN-based control networks in mobile machines



CAN-based control networks in mobile machines often have to combine different communication worlds, especially SAE J1939 and CANopen. This article provides an overview of used standards and specifications and explains how to interconnect the networks using CANopen and J1939 higher-layer protocols.

Source: Adobe Stock

Especially in mobile applications, system designers face the requirement that data has to be shared between multiple CAN networks – typically the networks for power train and the body application. The SAE J1939 protocol is the standard for the power train ECUs (electronic control units) in a vehicle, e.g., motor control or transmission. For trailer add-ons or special I/O requirements, CANopen modules are the first choice because of their broad functional range and availability (COTS: commercial off-the-shelf). This article provides an overview of used standards and specifications and explains how to interconnect the networks using CANopen and J1939 higher-layer protocols.

Application-specific solutions

For certain mobile applications, specific standards already exist that facilitate the system integrator's work.

The DIN 14700:2025-03 covers the integration for devices on fire-fighting trucks. The standard was originally developed by Rosenbauer, Magirus & Ziegler and presented as FireCAN in 2010. Over the past years, the original FireCAN standard has been refined very closely to the CANopen specification CiA 301 [1] and only the Legacy proprietary emergency messages remain.

Municipal vehicles can use the *Cle*ANopen CiA 422 application profile, which is based on the CANopen communication profile CiA 301. This profile is specifically designed to support applications such as waste collection vehicles and related municipal equipment. *Cle*ANopen allows the use of up to eight logical networks which can be assigned to different vehicle functions such as bin classification, lifting, weighing, compaction, and other operational tasks. To simplify integration and ensure standardized communication between devices, the profile uses pre-defined PDOs (process data objects) for real-time process data exchange and pre-defined SDOs (service data objects) for configuration and parameter access.

Specific CANopen profiles

The CiA 415 profile defines sensor systems for road construction and earth-moving machines. It supports the

integration of up to 126 sensor units, allowing a wide range of machine and environmental data to be collected and exchanged within the system.

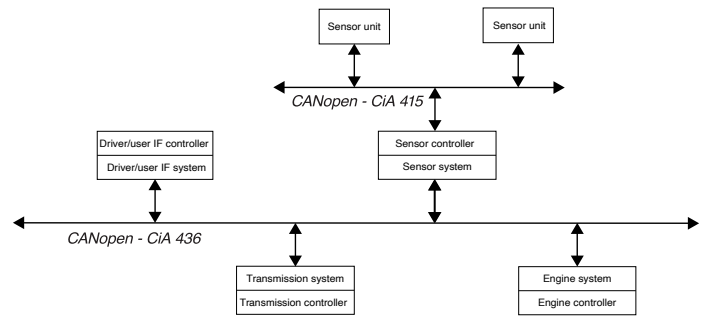


Figure 1: Structure of a CAN network according to CiA 415 and CiA 436 specifications (Source: CiA, Microcontrol)

The profile also provides an interface to CiA 436-4, which specifies a sensor system communication. Typical sensor data examples include machine speed, steering angle, material volume and mass, and environmental conditions. This standardized approach helps to improve interoperability between sensors, control units, and machine systems.

Interconnection via a gateway

The CiA 413 profile [2] defines CANopen-based truck gateway interfaces. Its main function is to connect a CANopen network with CAN-based in-vehicle networks used in trucks, buses, trailers, and other commercial or off-highway vehicles.

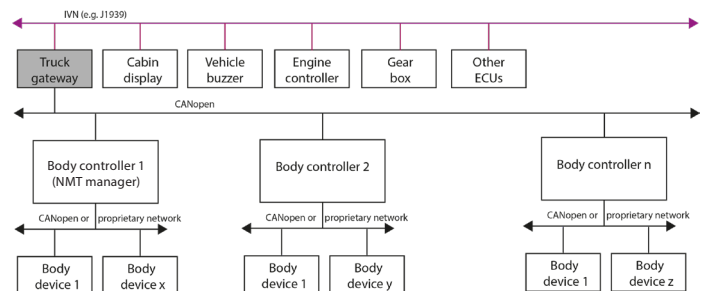


Figure 2: CiA 413 – Multiple-body controller system example (Source: Microcontrol)

CiA 413 enables communication between CANopen devices and vehicle networks based on SAE J1939, ISO 11992 truck/trailer networks, and related systems such as Isobus, NMEA 2000, or RV-CAN. The profile specifies standardized application objects for areas such as braking and running gear, vehicle equipment, superstructures, J1939-to-CANopen gateway communication, and HMI control (see Figure 2).

In practice, CiA 413 allows vehicle data and control signals to be exchanged in a structured way between the truck's in-vehicle network and external bodybuilder or superstructure systems. This helps to simplify integration, improve interoperability, and reduce the effort required to connect additional equipment to commercial vehicles.

Sharing the physical layer

The J1939-11 specification stipulates 250 kbit/s and is used in the majority of applications. The J1939-14 standard specifies 500 kbit/s for the physical layer. As a result, the bit rate for a shared physical layer is limited to the bit rates 250 kbit/s and 500 kbit/s. These two bit rates are also supported in CANopen, fortunately both standards define the same sample point location at 87,5 % together with a SJW (synchronization jump width) value of 1.

Beneath the physical layer examination, we have to take a closer look at the data link layer. The J1939-21 specification covers only extended frames, but allows the use of 11-bit identifiers from other protocols [3]. Mixed frames and extended frames have also an impact on the bus load and the priority. The IDE (identifier extension) bit is recessive for extended frames, so base-format frames have a higher priority and utilize the network first (see Figure 3).

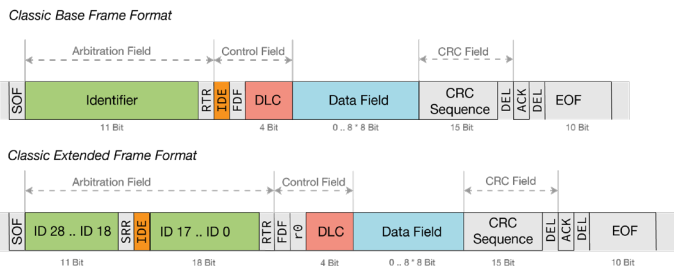


Figure 3: Classic base frame format vs. classic extended frame format (Source: Microcontrol)

CANopen “understands” J1939

Is it possible to use a CANopen device in a CAN network that runs with another protocol? The answer is: Yes, unless you keep two limitations in mind:

- ◆ The network shall not use the CAN-identifier value 0 (CANopen NMT: network management);
- ◆ The network shall not use the CAN-identifiers for SDO and NMT-EC services.

Both requirements are met by the J1939 protocol. But how can the CANopen device be set into the NMT state operational in a network without NMT manager? Simply by configuration of the object 1F80_h (NMT startup). Writing and storing a value of 8 to this object will tell the CANopen device to enter NMT state operational autonomously after

the NMT state initialization (self-starting device). The CAN trace in Figure 4 shows the required CAN frames to be sent (marked as Tx) to a device using node-ID 127.

No	DIR	ID (hex)	DLC	Data (hex)	Comment
001	Tx	67F	8	23 00 1F 00 08 00 00 00	SDO write, 1F80h
002	Rx	5FF	8	60 00 1F 00 00 00 00 00	SDO response, OK
003	Tx	67F	8	23 10 10 01 73 61 76 65	SDO write, 1010h, store config
004	Rx	5FF	8	60 10 10 01 00 00 00 00	SDO response, OK

Figure 4: CAN trace for configuration of a self-starting CANopen device (Source: Microcontrol)

In the next step, we need to configure the CANopen TPDO (transmit PDO) or RPDO (receive PDO) identifier for transmission or reception of a 29-bit J1939 protocol data unit (PDU).

J1939 recap – what is a PGN?

A J1939 PGN (parameter group number) identifies the content and function of a J1939 message. It is derived from the 29-bit CAN identifier, but it does not include the priority or source address. The relevant fields for PGN calculation are the Extended Data Page bit (EDP), Data Page bit (DP), PDU Format and, depending on the message format, the PDU Specific field.

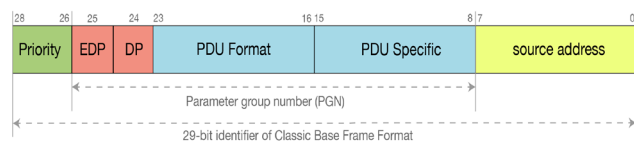


Figure 5: J1939 PDU identifier breakdown (Source: Microcontrol)

The PDU Format value determines whether the message uses PDU1 or PDU2 format. For PDU1, where the PDU Format is below 240, the PDU Specific field represents the destination address. Therefore, it is not part of the PGN and is set to zero for PGN calculation. PDU1 is mainly used for destination-specific communication.

For PDU2, where the PDU Format is 240 or higher, the PDU Specific field represents a group extension. In this case, it is a part of the PGN. PDU2 messages are typically broadcast messages.

In summary, the main difference is that PDU1 uses the PDU Specific field as a destination address, while PDU2 uses it as part of the PGN. This distinction is essential when decoding J1939 identifiers and mapping messages to the correct parameter groups.

Fake the J1939 protocol

As an example, we want to evaluate the J1939 Transmission output shaft speed (ETC1) message. The PGN for this message is 61442_d, it used a source address of 4 and a priority value of 3. This results in an J1939 PDU identifier value of CF00204_h. We want this identifier value to be evaluated by a CANopen receive PDO. The CAN trace in Figure 6 shows the required CAN frames to be sent (marked as Tx) to a device using the node-ID 127.

No	DIR	ID (hex)	DLC	Data (hex)	Comment
001	Tx	67F	8	23 00 14 01 04 02 F0 2C	SDO write, set CEFF, ID=0CF00204
002	Rx	5FF	8	60 00 14 01 00 00 00 00	SDO response, OK
003	Tx	67F	8	23 10 10 01 73 61 76 65	SDO write, 1010h, store config
004	Rx	5FF	8	60 10 10 01 00 00 00 00	SDO response, OK

Figure 6: CAN trace for configuration of a J1939 PDU identifier (Source: Microcontrol)

For device manufacturers using a CANopen protocol stack this configuration can be hard-coded, thus overwriting the CANopen pre-defined connection set as shown in Figure 7.

```
void CosPdoComSetup(void)
{
    //-----
    // Receive PDO communication parameter
    //-----
    atsRcvPdoComG[0].ulIdentifier = 0x0CF00204 | ID_MASK_IS_EXTENDED; // ETC1
    atsRcvPdoComG[0].ubTransType = ePDO_TYPE_EVENT_PROFILE;
}

```

Figure 7: Hard-coding of the J1939 PDU identifier for CANopen protocol stack users (Source: Microcontrol)

Faking the J1939 protocol with CANopen is perfect for PDU2 messages, the only downside is that the fake device lacks the J1939 address claiming and cannot respond to RQST (request) messages.

Devices supporting CANopen and J1939

A possible solution to overcome this disadvantage is to use devices supporting both protocols – CANopen and J1939. The CiA 510 specification defines how selected CANopen services are tunneled through J1939 networks. Its purpose is to enable CANopen-based device functionality to be integrated into J1939 systems without requiring a separate CANopen network.

For this tunnelling mechanism, J1939 provides the parameter groups PGN 1280, also known as CAM11, and PGN 1536, also known as CAM21. These PGNs are used to transport CANopen communication services, such as access to object dictionary entries, over a J1939-based network.

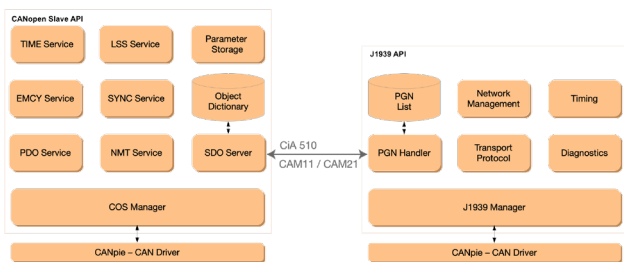


Figure 8: Tunnel between J1939 and CANopen (Source: Microcontrol)

This mechanism is used by profiles such as CiA 406-J and CiA 410-J, which specify the mapping of CANopen encoder and inclinometer profiles to J1939.

An increasing number of Microcontrol’s I/O modules supports this approach, giving the customer the free choice which protocol is used in the application.

Conclusion

CAN-based control networks in mobile machines often have to combine different communication worlds,

References

- [1] CiA 301, CANopen application layer and communication profile, <https://www.can-cia.org>
- [2] CiA 413, CANopen interface profile for truck gateways, Part 1 – 8, <https://www.can-cia.org>
- [3] SAE J193921, Data Link Layer, <https://www.sae.org>

especially SAE J1939 for powertrain and vehicle networks and CANopen for body applications, I/O modules, sensors, and special machine functions. Application-specific standards such as FireCAN (DIN 14700), CiA 422, CiA 415, and CiA 413 already provide proven solutions that reduce integration effort and improve interoperability.

Where CANopen and J1939 need to coexist, several approaches are possible. A shared physical layer can be used when bit rate and identifier usage are considered carefully. For selected applications, CANopen devices can be configured to evaluate J1939 messages directly by using extended PDO identifiers. However, this approach does not provide full J1939 functionality, such as address claiming or responses to request messages.

For more complete integration, devices supporting both CANopen and J1939 offer the most flexible solution. In this context, CiA 510 provides an important mechanism by tunnelling CANopen services through J1939 using CAM11 and CAM21. Together with mappings as specified in CiA 406-J and CiA 410-J, this enables standardized CANopen device profiles, such as encoders and inclinometers, to be used in J1939-based networks. As a result, system designers can select the communication approach that best fits the machine architecture while maintaining standardization, interoperability, and long-term scalability. ◀



Author

Uwe Koppe
 Microcontrol
koppe@microcontrol.net
www.microcontrol.net