Chapter 4
Higher Level Protocols

Gangolf Feiter, Lars-Berno Fredriksson, Karsten Hoffmeister, Joakim Pauli and Holger Zeltwanger

4.1 CANopen

CANopen is a standardized communication system that specifies communication profiles for the two lowest International Organization for Standardization (ISO)/ Open Systems Interconnect (OSI) layers as well as protocols for the ISO/OSI transport and application layers. Furthermore, CANopen provides device and application profiles in which transmitted process and configuration parameters as well as diagnostic information are specified.

The basic idea of CANopen was developed within an Elite Sport Performance Research in Training (ESPRIT) project. The registered association *CAN in Automation* (CiA) took over the first specification in 1994. Since then, the members of this international users and manufacturers association extend and maintain the CANopen specifications. Some of these specifications are freely available on the Internet, while the remaining specifications are only available for CiA members. Within the scope of this book, not all CANopen functionalities can be described. Further in depth going special literature is available.

G. Feiter (✉)
Concepts & Services Consulting, Alte Landstrasse 34,
52525, Heinsberg, Germany
e-mail: gangolf.feiter.csc@online.de

L.-B. Fredriksson
Kvaser AB, Aminogatan 25 A,
43153, Mölndal, Sweden

K. Hoffmeister
Elektrobit Automotive GmbH, Max-Stromeyer-Strasse 172,
78467, Konstanz, Germany

J. Pauli
Volvo Powertrain Corporation, Gropegårdsgatan,
SE-405 08, Göteborg, Sweden

H. Zeltwanger
CAN in Automation (CiA) GmbH, Kontumazgarten 3,
90429, Nuremberg, Germany

W. Lawrenz (ed.), *CAN System Engineering,* DOI 10.1007/978-1-4471-5613-0_4,

Table 4.1 Location of sampling point for several bit rates

Bit rate	Nominal bit time t_b (µs)	Valid range for location of sample point (%)	Recommendation location of sample point (%)
1 Mbit/s	1	75–90	87.5
800 kbit/s	1.25	75–90	87.5
500 kbit/s	2	85–90	87.5
250 kbit/s	4	85–90	87.5
125 kbit/s	8	85–90	87.5
50 kbit/s	20	85–90	87.5
20 kbit/s	50	85–90	87.5
10 kbit/s	100	85–90	87.5

Table 4.2 Maximal recommended bus and stub length

Bit rate	Bus length (m)	Stub length (max; m)	Accumulated stub length (max; m)
1 Mbit/s	25	1.5	7.5
800 kbit/s	50	2.5	12.5
500 kbit/s	100	5.5	27.5
250 kbit/s	250	11	55
125 kbit/s	500	22	110
50 kbit/s	1,000	55	275
20 kbit/s	2,500	137.5	687.5
10 kbit/s	5,000	275	1,375

Originally, CANopen was considered for usage in "embedded" networks in machine control systems. Meanwhile CANopen is used in a variety of industrial sectors as an "embedded" communication system. This also includes particular special purpose vehicles, trains, lifts and medical devices (e.g. computed tomography (CT) scanners) as well as ships.

4.1.1 Profiles for the Lower Layers

Currently, CANopen uses mainly the physical transmission according to ISO 11898-2. Alternative physical interfaces (ISO 11898-3 and Powerline) are being developed for specific applications. However, the bit timing is not explicitly specified in ISO 11898-1 and ISO 11898-2. Out of this reason, the CANopen communication profile (EN 50325-4 respectively CiA 301) defines sample points for several bit rates. Table 4.1 shows an overview of the bit timing parameters. Other bit rates are not allowed. Bit rates less than 50 kbit/s are not supported by all Controller Area Network (CAN) transceivers.

Table 4.2 shows the achievable bus length, on the basis of the recommended sample points, if direct current (DC) and alternating current (AC) voltage parameters for cables and connectors compliant to ISO 11898-2 are used. The actual

Table 4.3 CANopen pinning for the nine-pin DIN-Sub-D connector

Pin	Signal	Description
1	–	Reserved
2	CAN_L	CAN_L bus line (dominant low)
3	CAN_GND	CAN ground
4	–	Reserved
5	(CAN_SHLD)	Optional CAN shield
6	(GND)	Optional ground
7	CAN_H	CAN_H bus line (dominant high)
8	–	Reserved
9	(CAN_V+)	Optional CAN external positive supply (dedicated for supply of transceiver and optocouplers, if galvanic isolation of the bus node applies)

achievable bus length depends also on the internal delay times of the CAN controller and the CAN transceiver. For the exact design of the physical transmission, the formulae for line theory or appropriate empirical formulae should be used.

Basically, a linear bus topology with a 120-Ω termination resistor is required. Longer bus lengths may require the usage of a higher termination resistor. Not terminated bus lines are tolerable only to a certain degree, due to the reflections that could lead to a falsification of the bit value.

There are no detailed regulations on which bus cables or connectors should be used. However, there are recommendations for the pin assignments for a variety of connectors available. One of the widely used connectors is the nine-pin DIN-Sub-D connector (DIN41652). Table 4.3 shows the pin assignment given in the recommendation CiA 303-1.

CANopen uses the CAN protocol as described in ISO 11898-1. Although usage of remote frames is not recommended, they are basically allowed in some CANopen protocols. The various implementations of remote frames in CAN controllers could lead to problems when using CANopen protocols.

Some CANopen protocols use the base frame format (11-bit identifier) exclusively for data and remote frames, whereas other CANopen protocols allow the usage of the extended frame format (29-bit identifier). Error and overload frames are transparent for CANopen protocols. They are automatically processed by the CAN controller and lead, in case of an error frame, to an automatic retransmission of the interrupted frame or to a shutdown of the device (bus-off). Under rare conditions, it is to be kept in mind that a frame can be sent twice. Because of this situational condition, relative data or toggle commands shall not be transmitted.

4.1.2 Device Model

A CANopen-compatible field device features, from the hardware point of view, at least one CAN driver module and a CAN controller which implements the CAN protocol. Every CANopen device has to have its own object dictionary.

Table 4.4 Structure of CANopen object dictionary

Index range	Object
0000_h	Not used
0001_h–$001F_h$	Static data types
0020_h–$003F_h$	Complex data types
0040_h–$005F_h$	Manufacturer-specific complex data types
0060_h–$025F_h$	Device profile-specific data types
0260_h–$03FF_h$	Reserved
0400_h–$0FFF_h$	Reserved
1000_h–$1FFF_h$	Communication profile area
2000_h–$5FFF_h$	Manufacturer-specific profile area
6000_h–$67FF_h$	Standardized profile area 1st logical device
6800_h–$6FFF_h$	Standardized profile area 2nd logical device
7000_h–$77FF_h$	Standardized profile area 3rd logical device
7800_h–$7FFF_h$	Standardized profile area 4th logical device
8000_h–$87FF_h$	Standardized profile area 5th logical device
8800_h–$8FFF_h$	Standardized profile area 6th logical device
9000_h–$97FF_h$	Standardized profile area 7th logical device
9800_h–$9FFF_h$	Standardized profile area 8th logical device
$A000_h$–$AFFF_h$	Standardized network variable area
$B000_h$–$BFFF_h$	Standardized system variable area
$C000_h$–$FFFF_h$	Reserved

Table 4.5 Structure of the communication parameters

Index range	Object
1000_h–1029_h	General communication parameters
1200_h–$12FF_h$	SDO parameters
1300_h–$13FF_h$	CANopen safety parameters
1400_h–$1BFF_h$	PDO parameters
$1F00_h$–$1F11_h$	SDO manager objects
$1F20_h$–$1F27_h$	Configuration manager parameters
$1F50_h$–$1F58_h$	Parameters for program control
$1F80_h$–$1F91_h$	NMT parameters

Table 4.4 shows the structure of this parameter list. Table 4.5 shows the structure of the communication profile parameters. The single parameters of the object dictionary are addressable through a 16-bit index and an 8-bit sub-index.

If a field device implements several CANopen devices, it contains several object dictionaries as well. Additionally, it can provide bridge or gateway functionality. For example, the specification CiA 302-7 describes a CANopen-to-CANopen gateway with up to 32 CANopen devices, each having its own interface.

Three of the general communication parameters are mandatory:

- Device type (Object 1000_h)
- Error register (Object 1001_h)
- Identity object (Object 1018_h)

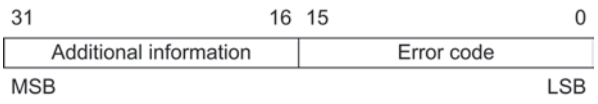


Fig. 4.1 Structure of the device-type parameter

Only the vendor ID has to be implemented in the identity object; all other parameters (product code, revision number and serial number) are optional. CiA uniquely assigns the vendor ID. Together with the other manufacturer-managed identity subparameters, every CANopen device can be addressed uniquely. The error register provides information if an error occurred in the device. The device type indicates which device or application profile is supported by the interface. The device-type parameter is a 32-bit value with the structure illustrated in Fig. 4.1. If the device does not support a standardized device or application profile, the profile number field shows a zero. If the CANopen device implements a standardized profile from CiA, it is shown through a number in the profile number field.

The value $FFFF_h$ in the additional information field indicates that this CANopen device supports several profiles. All other values are interpreted as profile specific.

The object dictionary provides space for eight logical devices (see Table 4.4). In case of several logical devices, the device type of the first logical device is implemented in index $67FE_h$. Each logical device has an 800_h address section available for profile parameters. The device-type object of the second logical device has the index $6FFE_h$.

Every logical device could contain several virtual devices. A minimum implementation of a virtual device may exist out of only one process date. More complex virtual devices comprise various process data and provides (if necessary) configuration data. The internal structure of a field device with several CANopen devices is illustrated in Fig. 4.2.

Every CANopen device must be assigned by the system developer with a unique 7-bit node ID. Because the node ID zero is reserved, a maximum of 127 devices is addressable in a CANopen network. The CAN identifiers, used in a variety of communication profiles to transmit and receive data frames, derive out of these node IDs. The setting and allocation of the node ID is not generally standardized in CANopen. The device manufacturer could, e.g. use a dual inline package (DIP) switch or provide an additional configuration interface. Another possibility is the coding of the connector: The device receives its node ID through an additional plug-in connection. In the CANopen profile for building door control systems, the patented node ID claiming procedure is used. When the CANopen interface is exclusively available, the Layer Setting Service (LSS), described in the specification CiA 305, can be used.

Every CANopen device must implement a communication state machine also referred to as network management (NMT) finite state automation (FSA) machine. Additional device-specific state machines could be necessary in the virtual devices. The NMT-FSA is part of the NMT, which is explained in the next subsection.

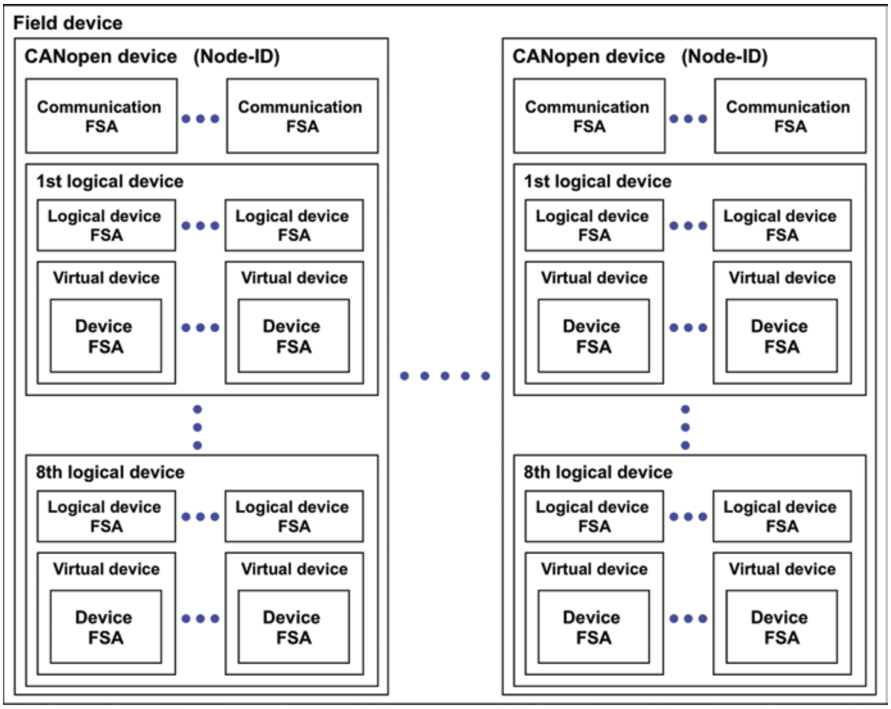


Fig. 4.2 Field device model

4.1.3 Network Management

The NMT is based on the master/slave principle. Only one active NMT master exists per CANopen network. The master controls all NMT-FSAs (referred to as NMT slave state machine) of the NMT slave devices. This also includes the FSA of its own CANopen NMT slave. To avoid misunderstanding, even the CANopen manager, which contains the NMT master, has to have an NMT slave state machine and an object dictionary. The so-called CANopen masters that do not have their own object dictionary are by definition not a CANopen device!

The NMT slave state machine supports four states (see Fig. 4.3): the volatile state "initialization" as well as the "pre-operational", "operational" and "stopped" states. The state transition from initialization to pre-operational takes place automatically. When the device reaches the pre-operational state, it starts its boot-up protocol, i.e. it sends its boot-up message, a single CAN data frame with a CAN identifier built-up out of the 4-bit function code (most significant CAN-ID bits) 1110_b as well as the 7-bit node ID (least significant CAN-ID bits). The boot-up message has a 1-byte data field that contains a zero. It is used to introduce the device after power-on or reset to the other members in the network. The same data frame is used in pre-operational and operational states to send the periodical *heartbeat protocol*. It contains

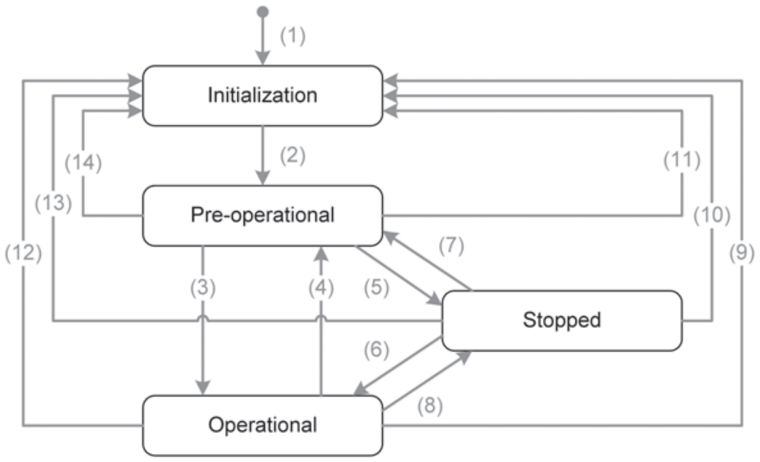


Fig. 4.3 CANopen NMT slave state

in 6 bits of the 1-byte data field the NMT state in which the device is situated at the moment (see Fig. 4.4). The frequency of the heartbeat could be configured in the *producer-Heartbeatheartbeat-time* (1017_h) parameter. If a CANopen device shall receive a heartbeat from another device, it has to be configured in the parameter set *consumer-heartbeat-time* (1016_h). Of course, the consumer time always has to be bigger than the corresponding producer time; otherwise, the device is always considered to be "lost".

If the consumer heartbeat time expires without reception of the corresponding heartbeat message, it will lead to an event (heartbeat event) in the heartbeat application of the device. The action is taken because this event is device or profile specific.

The state transitions are normally commanded by the CANopen device with NMT master functionality. For this purpose, the NMT master sends the CAN data frame with the highest prior identifier (CAN ID = 0). In this 2-byte message, the first byte contains the command (command specifier) and the second byte contains the node ID of the device that shall perform the state transition. In case of an error, the devices are able to change their states autonomously. The value 0 is interpreted as broadcast command, meaning all nodes have to perform this command. The executive state transition could be configured in the *error-behaviour* (1029_h) parameter.

For applications where no master/slave NMT is allowed due to safety issues, flying master solutions, as specified in CiA 302, are available. The protocols, specified in this standard, allow that if a device with NMT master functionality has a failure another device activates its NMT master functionality. In fact, it is possible that more than one device with "sleeping" NMT masters are located in the network. If the NMT master with the highest priority returns into a functional state, after a temporary failure, it could reclaim the NMT with the help of the specified protocols.

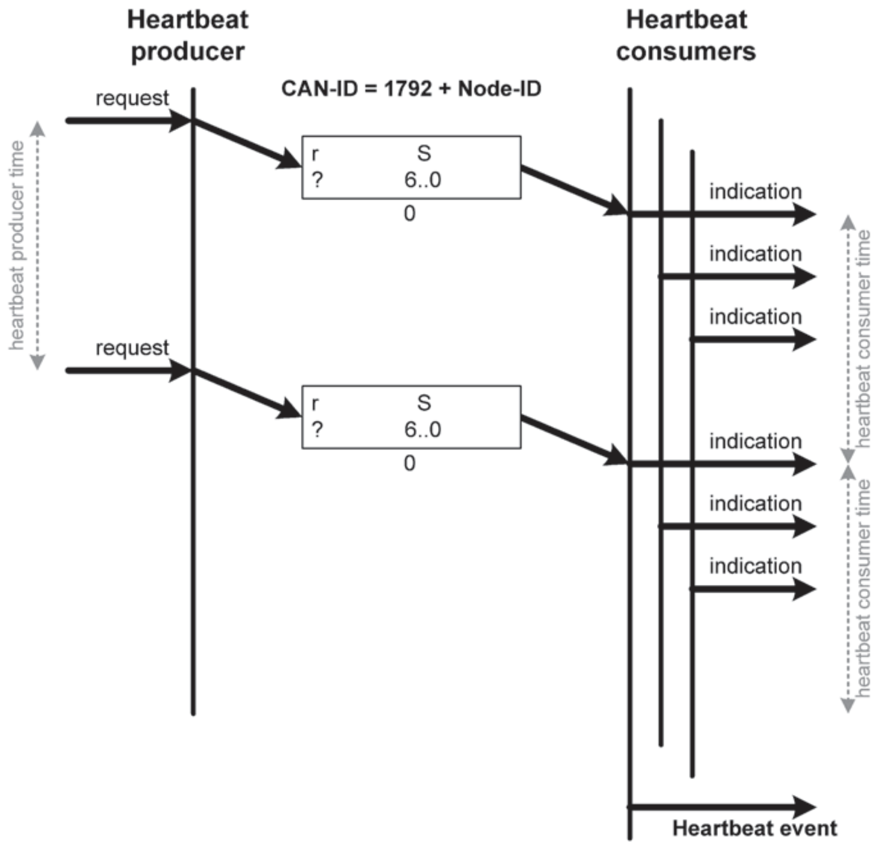


Fig. 4.4 Heartbeat protocol

4.1.4 Transport Protocols

The CANopen specification (EN 50325-4 respectively CiA 301) does not explicitly describe transport protocols. However, implicitly the service data object (SDO) protocols could be regarded as transport protocols in terms of the ISO/OSI reference model. The SDO protocols always work after the client/server model. The communication initiative always comes from the client and the server reacts to the "customers' wishes". SDO communication is allowed in pre-operational and in operational state. With SDO protocols, it is possible to write or read an object dictionary entry. Addressing takes place through the 16-bit index and the 8-bit sub-index. The two 8-byte data frames making an SDO are needed for: one, that the client could send his commands (command specifier) and if necessary the parameter data, and, the second one, so that the server could reply if the command was executed and could transmit in case of a read access the desired parameter data.

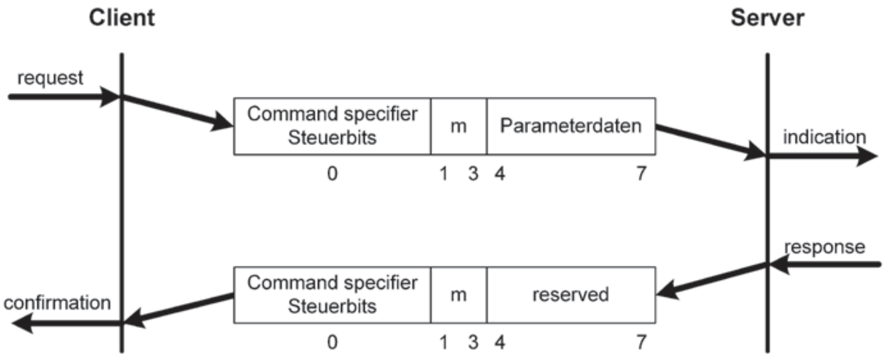


Fig. 4.5 SDO download protocol for not segmented data (m = index and sub-index)

Fig. 4.6 SDO upload protocol for segmented data

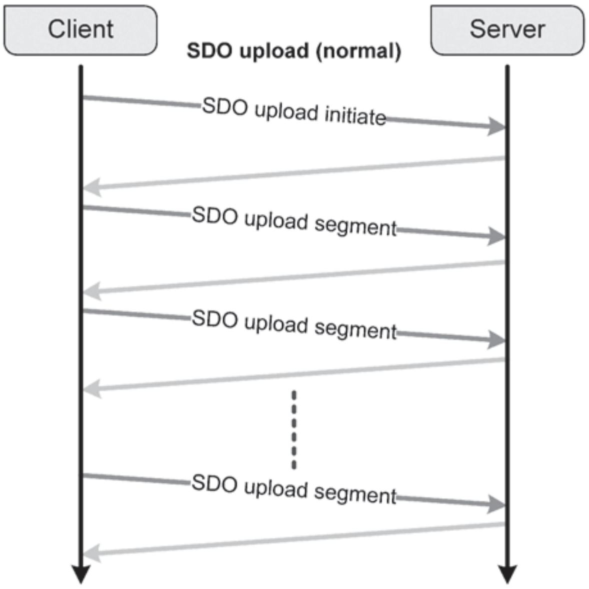


Figure 4.5 illustrates the protocol for parameter with a maximal length of 4 bytes (expedited SDO). The 8-byte data field consists of 1-byte command specifier, 3-byte index/sub-index and up to 4 bytes of parameter data.

If data, to be read or written, are larger than 4 bytes, they are segmented and transmitted with the same CAN data frames (segmented SDO) one after another (see Fig. 4.6). By doing so, every segment is confirmed. The first segment contains 4-byte parameter data. The following segments consist of up to 7-byte parameter data. Principally, the parameter data can have any length. An end-identification is sent in the command specifier of the last transmitted segment.

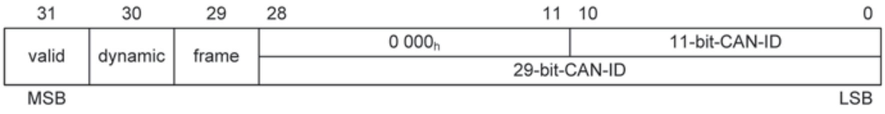


Fig. 4.7 Structure of the SDO COB-ID sub-parameter

To recognize if a segment was sent twice, every segment consists of a toggle bit. If a segment is sent twice, the server responds with an abort code. Both client and server are able to send an abort code in their SDO messages that aborts the current SDO communication, when problems occur.

Every CANopen device has to implement at least one SDO server to assure that at least one other CANopen device (normally the device with NMT functionality) is allowed to access (write or read) the object dictionary. The two needed CAN identifiers depend on the allocated node IDs: 1100_b + node ID for the to-be-received SDO message and 1011_b + node ID for the responded message. Every device is able to build up an SDO client and an SDO server relationship with all other devices in the CANopen network. Because there are no existing predefined CAN identifiers for these additional SDO connections, the connections have to be configured. Corresponding server and client SDO parameter sets are provided in the object dictionary (1200_h–$127F_h$ or 1280_h–$12FF_h$). Figure 4.7 illustrates the format of the configuration parameters for the CAN identifier. The 32-bit value consists of a valid byte for generating and deleting the data frame. The dyn-bit denotes if it is a static or a dynamic SDO connection. The frame-bit defines the frame format to be used and the CAN-ID bits with which the message shall be transmitted or received.

To improve the data throughput, the SDO block transfer exists. By using the SDO block transfer, only a configurable amount of segments, instead of all segments, are confirmed.

4.1.5 Application Protocols

Besides the NMT, boot-up, heartbeat and SDO protocols, CANopen also specifies protocols that uniquely assign the application layer of the ISO/OSI reference model. These include the process data object (*PDO*) *protocol* used for the transmission of time critical process data, the *emergency protocol* used for the notification of device and application failures, the synchronization (*SYNC*) *protocol* used for the synchronization of data capturing and data actuation as well as the *time protocol* used to set the system time.

Every CANopen device can transmit up to 512 PDOs and can receive up to 512 PDOs. PDOs are only allowed to be transmitted and evaluated during the operational state. PDOs use a CAN data frame and are sent according to the producer/consumer principle, which means there is always exactly one producer and one or more consumers of the CAN data frame. The complete 8 bytes of the data field

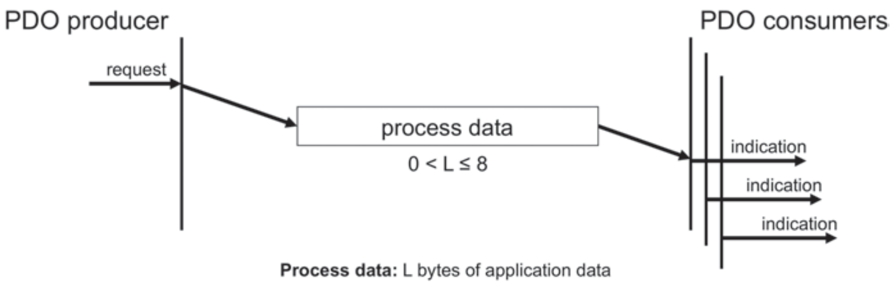


Fig. 4.8 PDO protocol

are available for the, to be transmitted, process data (see Fig. 4.8). Due to this the PDO does not need an overhead, in contrast to the CAN protocol, and therefore no additional bus bandwidth is necessary. The PDOs are configured with the help of two parameter sets: PDO communication parameter and PDO mapping parameter. The communication object identifier (COB ID), the transmission type, the inhibit time, the event timer and the SYNC start value belong to the PDO communication parameters. The 32-bit parameter COB ID consists of besides a few control bits, similar to the corresponding SDO parameter (see Fig. 4.7), the CAN identifier used for transmitting or receiving. The parameter transmission type defines how the PDO is to be transmitted or sent. The transmit PDO (TPDO) differs between a cyclic synchronous and an event-driven (asynchronous) transmission. When using the event-driven transmission, the event has to be defined in the device or application profile (transmission type: 255) or by the manufacturer (transmission type: 254). The event could be a temperature increase of 1°C or the change of a binary signal. The event could also be the expiration of the event timer. In this case, the PDO is sent periodically with the time (in milliseconds) configured in the event time parameter. If a signal-specific event and the event timer are defined, the PDO will be transmitted directly after the signal-specific event occurs. If no signal-specific event occurs, the PDO will be transmitted after the event timer has been elapsed. The event timer is restarted after every PDO transmission.

When a PDO gets transmitted cyclic synchronously (transmission type: 1), the process data are updated and transmitted after receiving a SYNC message. In order not to unnecessarily increase the busload with low-frequency signals, the PDO can be sent with only every second or 240th SYNC message (transmission type: 2–240). With the parameter SYNC start value, it is possible to configure the counter-value of the SYNC message (*sync counter*) to determine the first transmission. A possibility of reducing the busload lies in the usage of an acyclic synchronous transmission (transmission type: 0) in which the SYNC message has to be received and additionally a defined event has to occur. For receive-PDOs (RPDOs), a differentiation is only made with asynchronous or synchronous reception. With asynchronous reception the process data are immediately handled, whereas with synchronous reception the device waits for the next SYNC message to actuate the received data.

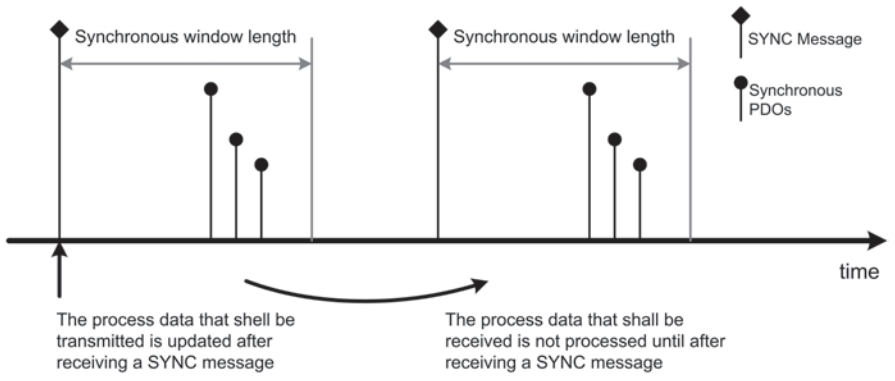

Synchronous window length
Synchronous window length
SYNC Message
Synchronous PDOs
time
The process data that shell be transmitted is updated after receiving a SYNC message
The process data that shall be received is not processed until after receiving a SYNC message

Fig. 4.9 Principle of the synchronous PDO communication

Figure 4.9 shows the synchronous transmission of PDOs. Synchronous PDO communication is especially common in the electric and hydraulic drive technology and in data acquisition of time-dependent linked values. The frequency of the SYNC message and the CAN identifier, on which the message is received or transmitted, are configurable via SDO.

Requesting PDOs via remote frames is possible in CANopen (transmission type: 252 and 253) but not recommended. Besides the additional busload, the variety of different implementations in the CAN controllers is problematic.

The inhibit time is the communication parameter in which the transmission of a PDO can be delayed for a certain time. With this method, the system developer can prevent the PDO from using the entire bus bandwidth. During the time period in which the highest priority PDO is not allowed to be sent, the second highest priority PDO becomes the highest priority PDO. With the usage of inhibit times, it is possible to achieve a completely deterministic behaviour of the PDO communication.

PDO mapping parameters determine what process data are to be received or transmitted via PDO. The PDO mapping parameters are defined by the device manufacturer or they are specified in the CANopen profiles. The entries in the mapping parameters contain pointers (index and sub-index) and determine which object dictionary entries are mapped into a PDO. Due to the fact that index and sub-index are local device addresses, the TPDO could contain pointers other than the corresponding RPDOs. The process data can be locally stored to different addresses in the object dictionary when more than one device receives the same PDO. Principally, the 8-byte data field of a PDO could be organized bitwise: This is the reason why a maximum of 64 mapping parameters is provided per PDO. If four 16-bit values are packed into a PDO, only four mapping parameters are needed. Simple CANopen devices only support static PDO mapping, i.e. the mapping set by the manufacturer is not configurable. When using variable mapping, the user is able to change the process data transmitted in the PDO, when the device is in pre-operational state. As a consequence, an optimization of the PDO transfer is possible. The user is able to group the needed process data for his/her application in one PDO. If the content of PDOs is changed when the device is in operational

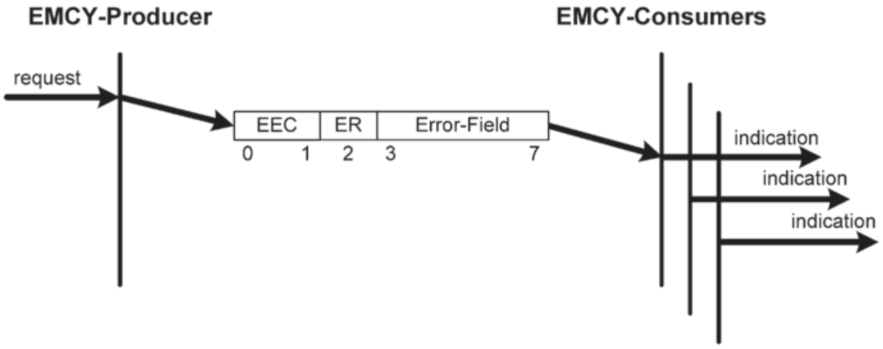


Fig. 4.10 Emergency protocol and structure of the message (EEC = emergency error code and ER = error register)

state (dynamic PDO mapping), the user has to pay attention to the data integrity of the PDO producer and consumer.

If the 512 TPDOs and the 512 RPDOs per device are not enough, it is possible to define a PDO as a multiplexed-PDO (MPDO). Similar to an SDO, it is only possible to transmit one process data in an MPDO. An MPDO contains the source or destination address of the object dictionary entry. Thus, a protocol overhead exists, but no confirmation about the reception of the MPDO is sent.

The emergency protocol is a special PDO that could be received by every other CANopen device if the emergency consumer parameter (1028_h) is configured. The emergency message consists of 8 bytes in which a standardized 2-byte emergency error code, the 1-byte error register and the manufacturer- or profile-specific 5-byte error field are transmitted (see Fig. 4.10). The CAN identifier, with which the emergency message is sent, is described in the emergency COB ID parameter (1014_h) and could be reconfigured if desired. The default value of the CAN identifier is 0001_b + node ID. It is reasonable to configure the emergency inhibit time (1015_h) to prevent an overload of emergency messages on the bus. In fact, it could happen that an error triggers a further error in another device, which could lead to a domino effect whereby many devices could start transmitting error messages over and over again.

The SYNC message is also transmitted after the producer/consumer principle. The SYNC message has no data field or a 1-byte data field containing the SYNC counter. The user can configure the counter (1019_h): 0 means that the counter-byte is not transmitted, and 2–240 represent the highest value that the counter supports. The other values are reserved. The SYNC message can be configured regarding the used CAN identifier, for transmitting and receiving (COB ID: 1005_h) as well as for the communication cycle period (1006_h) and the synchronous window length (1007_h) in microseconds. The synchronous window length serves as a measure for the producer and the consumer to indicate if a synchronous PDO was transmitted or received in the expected time. If the producer is not able to send the PDO (e.g. because bus access was denied), he could send an emergency message instead of the PDO. Same applies for the consumer: If a synchronous PDO is not received in

Table 4.6 Generic and application-specific CANopen device profiles

Number	Description	Status
CiA 401	Generic I/O modules	Free available
CiA 402	Drives and motion control	Only for members
CiA 404	Measuring devices and closed-loop controllers	Free available
CiA 406	Encoders (rotating and linear)	Free available
CiA 408	Fluid power technology	Free available
CiA 410	Inclinometer	Free available
CiA 412	Medical devices	Only for members
CiA 413	J1939-to-CANopen gateways	Only for members
CiA 414	Weaving machines	Free available
CiA 418	Battery modules	Free available
CiA 419	Battery charger	Free available
CiA 420	Extruder downstream devices	Free available
CiA 425	Medical add-on devices	Only for members
CiA 444	Crane add-on devices (e.g. spreader)	Only for members
CiA 445	RFID reader	Only for members
CiA 446	AS-Interface gateway	Only for members

the configured time, an emergency message is sent and the delayed reception will be ignored. The SYNC message is transmitted by default with the CAN identifier 128.

The time message is also based on the producer/consumer principle. The message contains a 6-byte value given in milliseconds after the 1 January 1984. The predefined CAN identifier is 256 and can be configured in the corresponding time COB ID (1012_h). With the time message, it is possible to do a network-wide time synchronization of the local timer units in the CANopen devices.

For transmission of safety-oriented information, special CANopen safety protocols, described in CiA 304, are available. The safety-related data object (SRDO) protocol uses two CAN identifiers that differ in at least 2 bits. Both frames are transmitted periodically, whereas the time interval between them is not allowed to exceed a certain value. The data of both frames are bitwise inverted and are double-checked by the consumers of the data. This concept of serial redundancy with integrated time expectation is able to detect all single faults. Due to this, the CANopen safety protocol is suitable for a safe data transmission up to Safety Integrity Level (SIL) 3 according to International Electrotechnical Commission (IEC) 61508.

4.1.6 Device Profiles

CANopen is one of the most standardized communication systems. This does not only apply to the communication protocols, specified in CiA 301 and CiA 302, but also apply to a variety of generic and application-specific device profiles (see Table 4.6). Device profiles specify the interface to the application program of a device in terms of process data and configuration parameter, which are normally writable and/or readable, in the range of 6000_h–$67FF_h$ in the object dictionary.

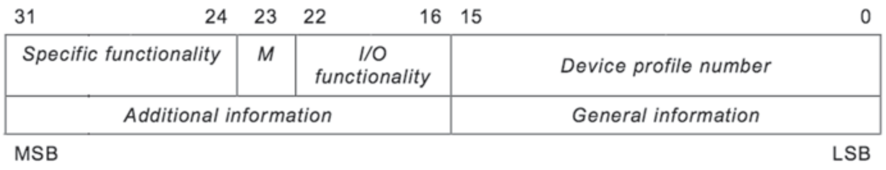


Fig. 4.11 Structure of a device-type parameter for CiA 401 modules

Table 4.7 I/O parameters for CiA 401 modules

Field name	Definition
Device profile number	401_d
I/O functionality—Bit 16	1^b=digital input(s) implemented 0^b=not implemented
I/O functionality—Bit 17	1^b=digital output(s) implemented 0^b=not implemented
I/O functionality—Bit 18	1^b=analog input(s) implemented 0^b=not implemented
I/O functionality—Bit 19	1^b=analog output(s) implemented 0^b=not implemented
I/O functionality—Bit 20 to Bit 22	Reserved
M(apping of PDOs)	1^b=device-specific PDO mapping is supported 0^b=predefined, generic PDO mapping is supported

Note: Any combination of digital/analog, inputs and outputs is allowed; one of the bits 16–19 shall be 1_b

The device-type object (1000_h) is described in detail in the device profiles. It does not only show which device profile is implemented but also show the available functions in the device. Figure 4.11 shows the structure of the device-type parameter for generic input/output (I/O) modules according to CiA 401 (Table 4.7).

The device profiles also determine the PDO communication and the PDO mapping parameters. The first four TPDOs as well as the first four RPDOs can each be assigned with a CAN identifier. The assignment, done in the device profiles, follows the same scheme as in other CANopen protocols. The four bits with the highest priority of the CAN identifier correspond to a PDO function (see Table 4.8), and the seven other identifier bits represent the node ID of the device.

This assignment guarantees that no CAN identifier is assigned twice. In other words, it is not possible that two CANopen devices send data frames with the same identifier leading to a not solvable bus access conflict. The CANopen device with the NMT master functionality is normally a programmable controller that has the corresponding RPDOs and TPDOs. Thus, per default there is only one master/slave relation concerning PDOs. If a system developer wants to realize PDO cross-communication or wants to implement more than one controller in the network, the configuration of PDO identifiers is necessary. This procedure is also known as PDO linking and is supported by software tools. In this way, it is even possible to realize distributed PLC systems.

Table 4.8 Default CAN identifiers for TPDOs and RPDOs

Name	Function code	Resulting CAN-IDs
TPDO1	0011_b	385 (181_h)–511 ($1FF_h$)
RPDO1	0100_b	513 (201_h)–639 ($27F_h$)
TPDO2	0101_b	641 (281_h)–767 ($2FF_h$)
RPDO2	0110_b	769 (301_h)–895 ($37F_h$)
TPDO3	0111_b	897 (381_h)–1023 ($3FF_h$)
RPDO3	1000_b	1025 (401_h)–1151 ($47F_h$)
TPDO4	1001_b	1153 (481_h)–1279 ($4FF_h$)
RPDO4	1010_b	1281 (501_h)–1407 ($57F_h$)

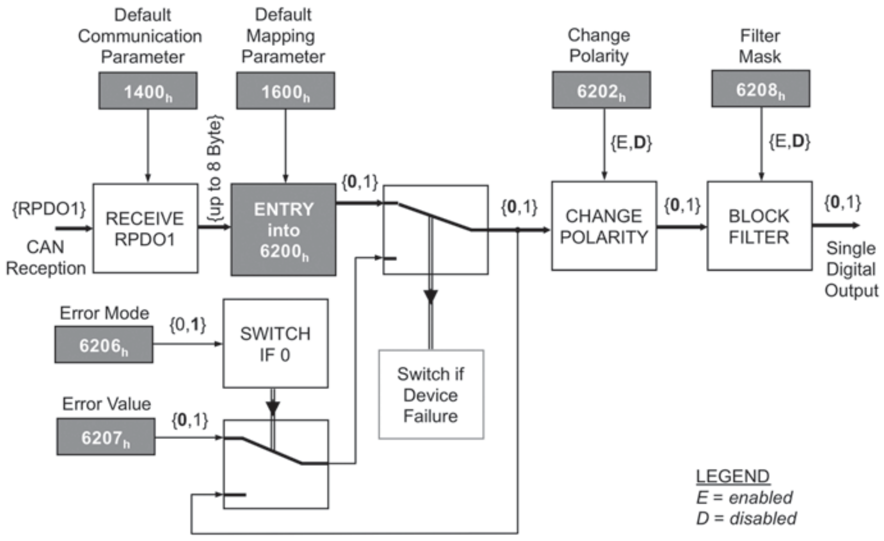


Fig. 4.12 Block diagram of a digital output module according to CiA 401

The most frequently implemented device profile is the profile for generic I/O modules (CiA 401). It supports digital and analog inputs and outputs. The user is able to parameterize the modules in a standardized way regarding the I/O functionality. With the parameter set "error value output (6207_h)", it is possible to set the value of the digital outputs that should be used, when an internal device error occurs. Another example is the parameter set "analog input interrupt trigger selection (6421_h)". By this parameter set conditions can be defined under which a PDO transmission is triggered by an analog input, which has been mapped into a TPDO. These conditions can be exceeding or falling below a certain threshold or the predefined change of a given value. It is also possible to set the limit and delta values under a fixed address (index and sub-index) in the object dictionary.

Figure 4.12 shows the block diagram of a digital output module, and Fig. 4.13 shows the block diagram of an analog input module. The device profile CiA 401

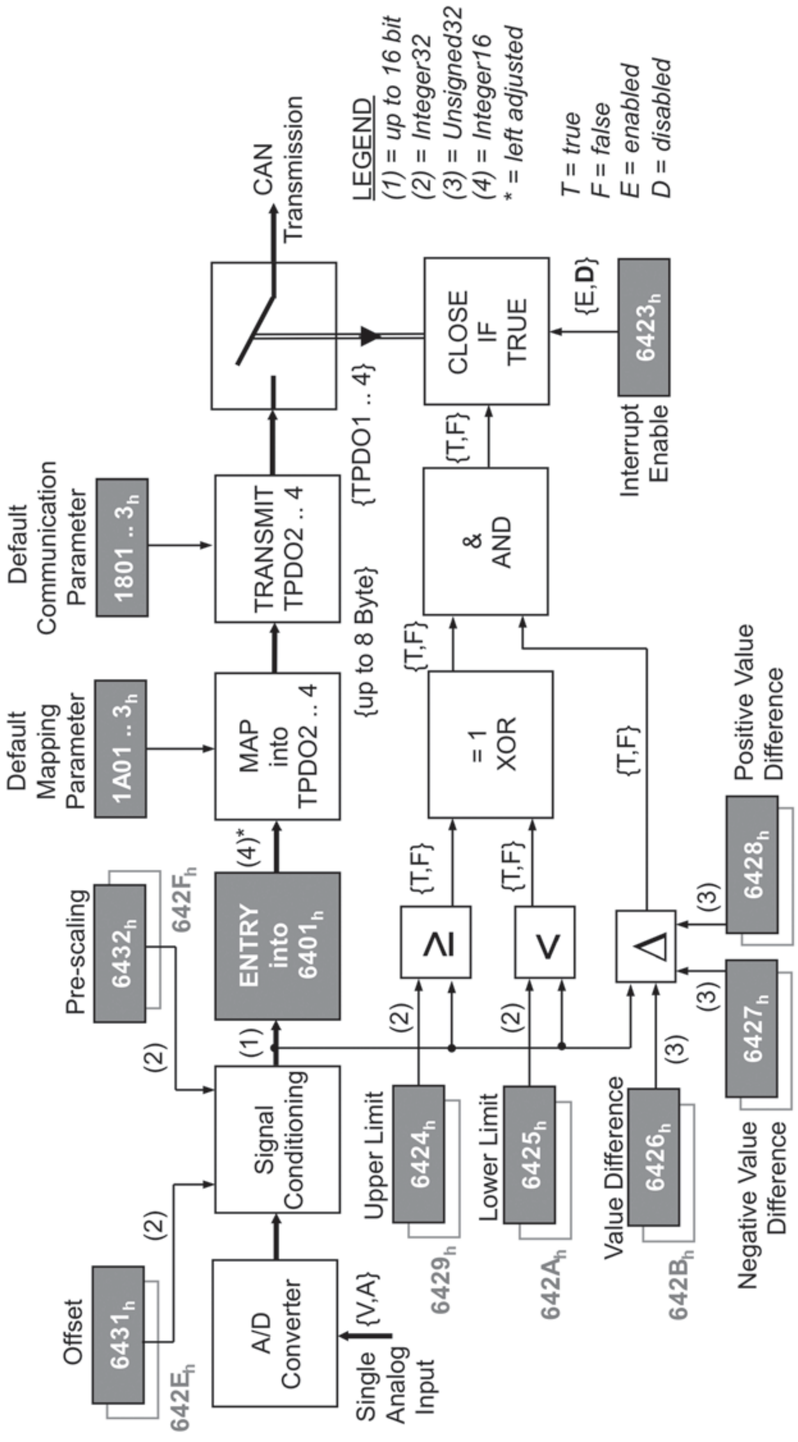


Fig. 4.13 Block diagram of a digital input module according to CiA 401

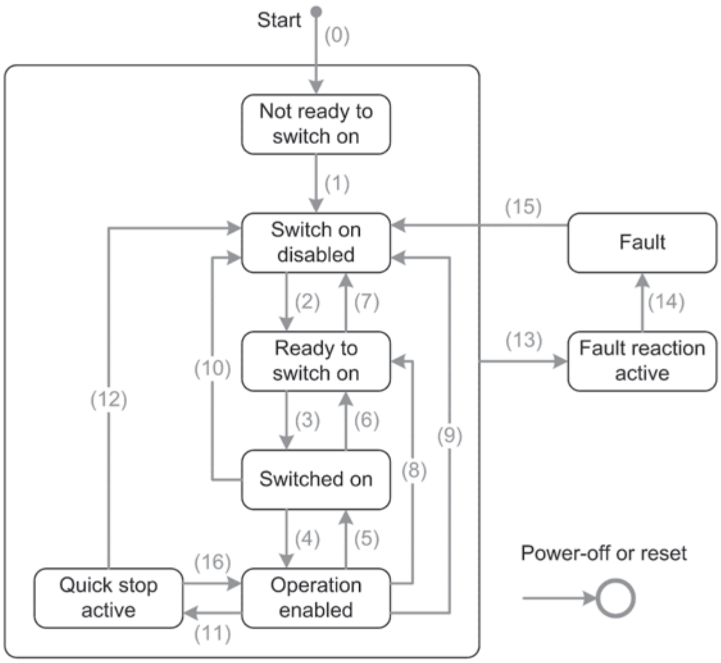


Fig. 4.14 State machine of an electrical drive according to CiA 402

also defines a default PDO mapping. According to the defined PDO mapping, 8×8 digital input signals are located in the first TPDO and 4×16-bit analog input values in the following three TPDOs. In the first RPDO 8×8 digital output signals and in the three following RPDOs 4×16-bit digital output values are located. Further digital and analog I/O signals could be placed in manufacturer-specific PDOs. All default PDOs are valid and have a predefined CAN identifier. They are transmitted event driven (transmission type: 255). The event is a configured trigger condition. Furthermore, all TPDOs that are not switched off generally are transmitted when a transition from pre-operational state into operational state occurs.

The device profile CiA 401 supports a digital granularity of 8 bytes per default.

1-bit, 16-bit and 32-bit accesses are optionally provided. Analog modules have a 16-bit resolution specified per default. Alternatively, analog 8-bit values, 32-bit values as well as floating-point arithmetic and manufacturer-specific analog formats can be implemented.

Some of the CANopen device profiles also specify application-specific state machines. The state machine, shown in Fig. 4.14, is described in the drives and motion control profile CiA 402 (see IEC 61800-7). It is controlled by a control word (6040_h) received via RPDO. After a state transition, the CANopen device transmits the status word (6041_h) to the host controller for purpose of acknowledgement on the application level.

In the drives and motion control profile, all necessary parameters for operation are listed in the object dictionary as defined addresses (index and sub-index). Even setpoints or actual values are transmitted via PDOs. The drives and motion control profile CiA 402 is suitable for simple frequency converters as well as for complex servo controller and stepper motor controls.

For some industries, specific device profiles were developed. Such profiles simplify in particular system integration in modular mechanical engineering. The system developer does not have to link generic digital and analog inputs and outputs together but is able to integrate a complete subsystem on a higher level. Typical examples are collimators and dosimeters in the field of medical devices, thread-feeding equipment in weaving machines or spreaders in crane systems. The members of CiA will keep publishing generic and branch-specific device profiles in the future.

4.1.7 Application Profiles

With the approach of the device profiles, the system developer is allowed to integrate devices from various manufacturers in a CANopen network with one or several programmable controllers. By doing so, the plug and play functionality is limited to a master/slave relation concerning the PDO communication. Although it is possible to configure a PDO cross-communication between any of the devices, cross-communication is not predefined. Out of this reason, branch-specific application profiles were made particularly to meet these requirements.

Application profiles describe all interfaces of a CANopen application in an object dictionary. The object dictionary entries have identical meaning in all devices. Application profiles specify functional units, also referred to as virtual devices. A CANopen device is able to integrate up to eight application profiles (each located in a logical device). Each logical device, on the other hand, contains a number of virtual devices. The information on what virtual devices are implemented could be found in the device-type parameter (1000_h) or in an application profile-specific parameter. With the concept of virtual devices, it is possible to describe transparent CANopen bridges/gateways in an easy way.

Furthermore, the concept of the application profiles opens the possibility of implementing, in an extreme case, only one virtual device in a single CANopen device and all other virtual devices in another CANopen device as well as the combination of both of these devices through a CANopen network. It is also possible to implement every virtual device in a CANopen device and let them communicate through CANopen. It is not possible to spread a virtual device over more than one CANopen device. Out of this reason, it is very important to ensure, when creating an application profile, that the granularity of the virtual devices is suitable for future requirements. Table 4.9 shows the already published application profiles and the application profiles that are still in development.

The application profile for lift control systems (CiA 417) specifies several virtual devices (see Table 4.10). The object dictionary (range 6000_h–$9FFF_h$) is divided into

Table 4.9 CANopen application profiles

Number	Description	Status
CiA 415	Road construction machine sensors	Only for members
CiA 416	Building door control systems	Only for members
CiA 417	Lift control systems	Free available
CiA 421	Train vehicle control networks	Only for members
CiA 422	Municipal vehicles	Only for members
CiA 423	Rail vehicle power drive systems	Only for members
CiA 424	Vehicle door control systems	Only for members
CiA 426	Exterior rail vehicle lighting	Only for members
CiA 430	Auxiliary rail vehicle lighting	Only for members
CiA 433	Interior rail vehicle lighting	Only for members
CiA 436	Construction machineries	Only for members
CiA 437	Grid-based photovoltaic systems	Only for members
CiA 447	Special-purpose car add-on devices	Only for members
CiA 455	Drilling machines	Only for members

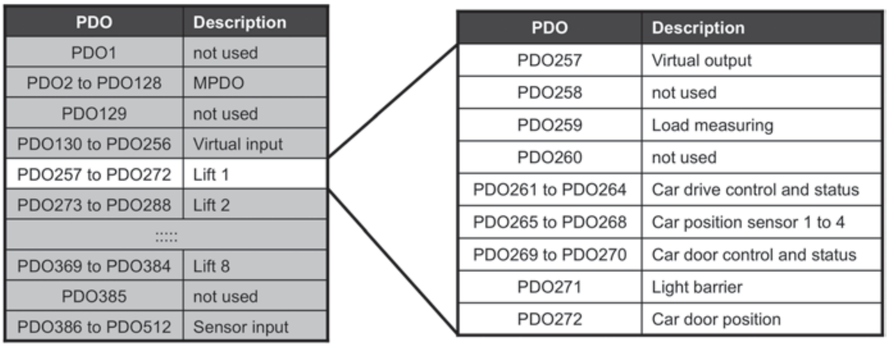
Table 4.10 Virtual devices of the CANopen lift application profile

Virtual device	Function
Call controller	Receives all call requests
Input panel unit	In-car call panel or floor call panel
Output panel unit	In-car display panel or floor display panel
Car door controller	Transmits commands to the car door unit
Car door unit	Cabin doors
Light barrier unit	Light barrier in the cabin doors
Car position unit	Position measurement of the car (according to CiA 402)
Car drive controller	Transmits commands to the car drive unit
Car drive unit	Drive unit (according to CiA 406)
Load measuring unit	Measuring of current load of the car
Sensor unit	Glass breakage, smoke, pressure, temperature sensor, etc.

800_h segments. Every segment could represent a lift control, i.e. it is possible to describe a group control containing up to eight lift shafts with this profile. Every virtual device has several configuration parameters that are readable or writable via SDO as well as process data, that is transmitted or received, in PDOs.

Every CANopen device according to CiA 417 supports a Transmit-MPDO and up to 127 Receive-MPDOs as well as, depending on the implemented virtual device, further dedicated TPDOs and RPDOs. Due to the fact that all PDOs have an assigned CAN identifier (see Fig. 4.15), the system developer must not assign any CAN identifiers. However, if the devices support PDO linking, he/she is able to optimize the PDO communication in his/her lift application with regard to priority and data content.

The CANopen lift application profile supports up to 254 floors per lift control. In total, 32 times 254 digital inputs per lift control are addressable. When using eight lift controls, a total of 65,024 digital inputs are available.



PDO	Description
PDO1	not used
PDO2 to PDO128	MPDO
PDO129	not used
PDO130 to PDO256	Virtual input
PDO257 to PDO272	Lift 1
PDO273 to PDO288	Lift 2
:::::	
PDO369 to PDO384	Lift 8
PDO385	not used
PDO386 to PDO512	Sensor input

PDO	Description
PDO257	Virtual output
PDO258	not used
PDO259	Load measuring
PDO260	not used
PDO261 to PDO264	Car drive control and status
PDO265 to PDO268	Car position sensor 1 to 4
PDO269 to PDO270	Car door control and status
PDO271	Light barrier
PDO272	Car door position

Fig. 4.15 PDO assignment in a CANopen-lift network system compliant to CiA 417

It is possible to implement up to eight application profiles in a single CANopen device, if the object dictionary entries of a logical device are used for the application profile. This possibility could be used in the application profiles for train vehicle control systems. From a logical point of view, it is a hierarchically arranged virtual network that can be mapped one-to-one on the physical CANopen network. However, it is also possible to represent up to eight virtual networks on a CANopen interface. The implementation flexibility of the application profiles allows the possibility of using the same specification in a simple and in a complex system. The “bridges” needed for the implementation of several physically separate CANopen networks are PDO transparent, i.e. the PDOs are just reached further due to their system-wide validity. If it is wished to configure the entire system from one single point via SDO, it is necessary to implement CiA400, which enables remote SDO communication. It is therefore necessary to assign every network with a unique network ID.

4.2 AUTOSAR

4.2.1 Introduction

4.2.1.1 AUTOSAR Foundation

“AUTOSAR” is the abbreviation for “AUTomotive Open System Architecture”. The organization of the AUTOSAR standard was founded in 2003 as a development partnership of international automobile manufacturers and supplier industry. The goal is to develop a standardized software architecture and standardized software interfaces for automotive electronic systems.

The development of such standards for software development in the automotive industry was long overdue. The software functionality within vehicles increases a lot and the complexity is growing. At the same time there is a high demand to deliver high-quality software, partly also due to the growing amount of functional safety-related systems. Some of the original equipment makers (OEMs) react to this global trend with own standardized software platforms, for example, the BMW *Standard Core*.[1] This software platform was provided by BMW to the Tier 1 already in 1998. Because of the existence of different OEM-specific automotive middleware, the Tier 1 always needs to adapt their application software to these quasi-standards. The integration costs of these adaptations are sometimes beyond the application development costs. In parallel to these technical requirements, there is a change in the market environment. The major sales regions are saturated. This forces the OEM to change their strategy for vehicle production. In addition, niche markets will be addressed with special vehicle models and the model change will be faster. The pressure for vehicle development, and hence also for automotive software development, increases.

The only way to develop more complex system in a shorter time—and still control them—is to standardize a horizontal basic software while reusing the applications more and more. A first step to the right direction is the AUTOSAR standard.

The following pages will give a brief overview about the AUTOSAR standard by focusing the CAN-based communication path through the layered software architecture.

4.2.1.2 AUTOSAR Concept

One of the main ideas of AUTOSAR is the clear separation of hardware-dependent software from hardware-independent software. Therefore, an abstraction layer is placed between the microcontroller hardware and the application software. The AUTOSAR runtime environment (RTE) is on top of the layered architecture. It is the only visible interface from the application point of view. Below the RTE several software layers exist, from services via operating system (OS) down to the lowest software layer, the microcontroller abstraction layer (MCAL). Figure 4.16 shows an AUTOSAR compliant software architecture.

The AUTOSAR RTE defines the interface from the application to the basic software. For data exchange, the RTE implements, e.g. a client–server and/or a sender/receiver communication model.

All interactions of application software, called "application software components" in the AUTOSAR language, run via the RTE. This results in a hardware-independent application. This enables the exchange of applications among the different electronic control units (ECUs). If there is a lack of resources on one ECU, e.g. no more random access memory (RAM) left to run a special RAM-intensive

[1] *Standard Core* is a typical name for a complete automotive middleware respectively for the software between application and hardware of an electronic control unit (ECU). Additional to the pure runtime software a Standard Core contains support functions like a generic make environment or complex configuration tools.

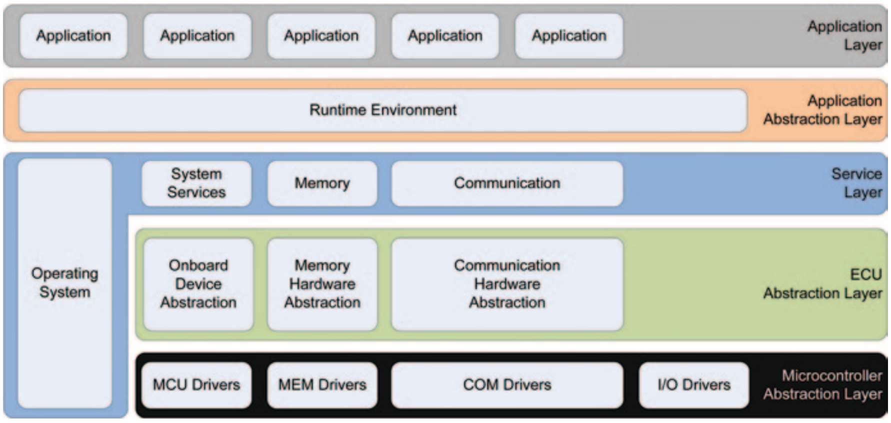


Fig. 4.16 The layered architecture of an AUTOSAR Standard Core. (Elektrobit Automotive GmbH, AUTOSAR—Getting Started)

application algorithm, it can be executed on another ECU if the input and output signals of these application software component are also available on the other ECU.[2] With this independence from hardware it is in addition possible to reuse the application software component on another hardware architecture, e.g. on the next generation of the ECU. Software will become also a "carry-over part" for the automotive development.

The basic software itself is also structured in hardware-dependent and hardware-independent software parts. The only real hardware-related software part is the OS and the MCAL—all other software can be reused, too. With this portability major parts of the basic software can become a carry-over part with all the benefits of reused standard software.

4.2.1.3 New Methodology

After the successful separation of application software and hardware-dependent software, new methodologies within the automotive vehicle development can be applied. Function-oriented development will replace ECU-oriented development. AUTOSAR defines a complete methodology for the function-oriented development. The methodology is not a complete process description, this was never the goal of AUTOSAR, but it gives a brief structure in the overall development process. In brief, the AUTOSAR methodology starts with the pure software functions, which are represented by the application software components. The inputs and outputs of these functions, the signals, are defined and the connections are known. All this information is collected in a so-called "System Configuration Description", which

[2] And the timing requirements to the signals are fulfilled (e.g., availability and, maximum jitter).

is an Extensible Markup Language (XML)-based data structure defined by the AUTOSAR meta-model. The system configuration description contains the description of all software components, the ECU resources and the system constraints for a complete vehicle. The next step, after the definition of many system parameters, the configuration of one ECU is extracted from this global vehicle description. This extract of the system configuration forms together with the configuration of the basic software the so-called ECU configuration description. Within this ECU-specific configuration, the exact definition, e.g. of OS tasks is done. System configuration and ECU configuration are two processes with high interaction and are done usually by two different developer groups.

With AUTOSAR, the role model of the automotive industry might change. The classic supplier pyramid will be replaced by a network of suppliers because the tasks, especially the software related, can be done in future a little bit more independently and in parallel. The software companies will play an important role; they provide the software platforms where all the applications will run on top. In addition, the same or different software suppliers can provide pure software functions as a product. If the standard is established, these software can be used in several cars, independent of the OEM.

4.2.2 The AUTOSAR Platform

4.2.2.1 Overview

The legacy software platforms from each OEM differ in interfaces and content. The goal of the AUTOSAR standardization is to form a universal standard which need not have to be adapted for each OEM.

The AUTOSAR specifications deal with many software modules, needed for an ECU development. Old standards, like the OSEK/VDX "Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug / Vehicle Distributed eXecutive" standard, comprised less functionality, e.g. the mentioned OSEK standard only defines an OS, a communication model and NMT. The implementation of these standard modules found their way into many cars of many OEM, but AUTOSAR is going beyond; Fig. 4.16 shows a much broader approach and shows much more standardized modules of the AUTOSAR-layered architecture.

In principle, every module can be assigned to one of the four layers: the application abstraction layer, service layer, ECU abstraction layer and MCAL. The application abstraction layer forms the only interface to the applications. The implementation of this layer is the RTE, and the RTE is the "glue code" which connects the applications to the lower layers. The service layer implements system services, memory access services, communication services and the OS services. The ECU abstraction layer is used to implement an abstraction of the peripheral I/O, memory (e.g. flash memory or electrically erasable programmable read-only memory (EEPROM)) and communication. The services are based on this abstraction layer and the abstraction layer itself connects, e.g. different communication channels by using the lowest layer, the MCAL. In this lowest layer, the drivers for accessing the hardware are implemented. This lowest layer has to be developed for each new microcontroller—it is the only piece of software which is not hardware independent.

4.2.2.2 Microcontroller Abstraction Layer

As mentioned above, the MCAL is the only layer which depends on the hardware. The above layers have no dependency any more, except the OS. This principle is the basis for an easy adaptation of the basic software to new hardware architecture or a new derivative; only this deepest layer will be exchanged. The layers above will be tested together with the exchanged lower layers in an integration test system. There are exceptions within the higher levels of software if special hardware features shall be used there, too. If these features are not able to be implemented only in the MCAL, a hardware dependency is placed in the higher level software—the supplier of this basic software has to check if this dependency is maintainable and/or if the costs for hardware dependency are less than the, e.g. speed advantage gained by using special hardware features.

The drivers, of the MCAL, can be subdivided into the following four groups:

- Microcontroller drivers—they contain software to control the microcontroller core (Micro-Controller Unit (MCU) driver), timer modules (General Purpose Timer (GPT) driver) and on-chip watchdog (watchdog driver).
- Memory drivers—these are the drivers to access EEPROM and FLASH memory.
- Communication drivers—these are used to connect bus systems like Serial Peripheral Interface (SPI) bus for inter-ECU communication, e.g. to connect an external SPI EEPROM device, and to access common used bus systems like CAN, Local Interconnect Network (LIN), FlexRay and also Ethernet.
- I/O drivers—these drivers connect the on-chip peripherals like digital I/O, analog digital converter (ADC), input capture unit, pulse width modulation (PWM) and the port driver for the configuration of each port pin functionality.

Within the AUTOSAR architecture, hardware access will be enabled by two software modules: the driver of the MCAL and the corresponding interface of the layer above—the ECU abstraction layer.

4.2.2.3 ECU Abstraction Layer

For almost each driver of the hardware-dependent layer exists an interface to encapsulate the lowest layer. The access to this interface layer is always identical for all layers above. For example, multiple CAN modules can be summed-up abstract by the interface layer, e.g. by controlling several CAN drivers. The AUTOSAR COM module in the service layer can access the CAN interface always in a same way, sending only a protocol data unit (PDU)[3] to the interface. The interface is

[3] PDU: protocol data unit.

configured to forward the PDU to the dedicated bus system—in this example, to a dedicated CAN bus. An exception to this driver/interface relationship is the group of the I/O driver. There exists no interface layer for the I/O drivers—instead there is an "I/O hardware abstraction" module to implement the bridge between the low-level hardware-dependent driver and the highest layer of the application abstraction layer. The specification of this I/O hardware abstraction is more like a guideline for the implementation than a real implementation with application program interface (API) specification, etc. The reason is the various possibilities of connecting I/O to a microcontroller; hence, the implementation of this I/O hardware abstraction is in many cases project specific and peripheral specific. There could be, for example, power switches which need already a combination of two I/O (ADC plus PWM) or an algorithm implemented in this I/O hardware abstraction (e.g. debouncing of digital input).

4.2.2.4 Service Layer

The service layer of an AUTOSAR-based software architecture provides many services which can be accessed from the applications via the RTE. The modules of this layer are complex state machines, e.g. the ECU state manager which controls the state of the ECU like OFF, RUN and SLEEP. The module groups of the service layer are as follows:

- State manager—it manages all the states of an ECU, communication and, e.g. of the Watchdog.
- Memory manager—it controls the access to persistent memory. The non-volatile RAM manager (NVRAM-Manager) is a core module of the memory services. This module manages the non-volatile memory blocks of an EEPROM, a FLASH–EEPROM emulation or other external memory devices. The tasks are, for example, write, read access, initialization of memory-mapped blocks and check if the cyclic redundancy check (CRC) is still valid.
- Vehicle communication services—services of this service module group enable the communication across the border of the ECU. The lower layers support the known protocols like CAN, LIN and FlexRay—in future also Ethernet. The jobs of these services are the abstraction of the access via standardized interfaces and data encapsulation via abstract PDUs instead of protocol-specific messages, control of the network by support of NMT services and access to diagnostic services.
- OS services—the OS provides several services. For example, the management of the central processing unit (CPU) time for tasks and interrupt service routines. There are event mechanisms and protection mechanisms for time and memory access if provided by hardware. The memory protection is only possible if the controller has a memory management unit (MMU) or memory protection unit (MPU). The AUTOSAR OS also provides mechanism for time synchronization—e.g. to synchronize the internal schedule with an external clock, provided by FlexRay.

4.2.2.5 Application Abstraction Layer

The highest layer of the AUTOSAR-layered architecture is the interface to the application. This application abstraction layer is the only interface of an AUTOSAR application to the AUTOSAR basic software. No AUTOSAR application shall access functions from the basic software direct. Out of the application perspective, there are no tasks or interrupts anymore and basic software interfaces do not exist. This highest layer will be implemented only by one software module: the AUTOSAR RTE.

The RTE implements the connection of all communication paths and the abstraction of the runtime management. An AUTOSAR compliant application—application software component—implements a software function. This application software component consists out of a formal description in XML, the Software Component Description (SWC-D) and the implementation in C source code. The RTE itself will be generated with XML input data of the ECU configuration and all the application software component descriptions. The API of the RTE depends on the software components, their signal names and internal structure. The RTE is the implementation of the virtual function bus (VFB), which is the core element of the AUTOSAR architecture.

The VFB is the communication medium for all functions and provides sender/receiver or client/server communication methods. In the world of the VFB, it is not relevant where the function is executed and how the data are transported. The "mapping" of functions to ECUs is part of the system design process—the result of this process is the system description. The path for the signals will be defined by the distribution of the application software components to real physical ECUs. For the application it is irrelevant how the data are transported; this will be done by the configured basic software and the RTE. A signal could be a direct result of a digital input line or it could be a part of a CAN message (Fig. 4.17).

4.2.2.6 AUTOSAR Applications

AUTOSAR compliant applications are connected via the RTE to the basic software platform. The API of the RTE provides many possibilities of designing an application. In principle, the communication is running via "ports" and the activation of runtime parts is done by the execution of the so-called "runnable entities". The communication ports can be of the type sender/receiver or client/server. The client/server type represents the typical function call, which is needed to call, e.g. library functions (e.g. mathematical functions). As already mentioned, the information from where the RTE gets the data is part of the RTE configuration (which is part of the ECU configuration). With this method it is possible to develop application software and define later the final communication path for each input or output signal. This is the basis for shifting application functions within the vehicle ECU topology—one of the goals of AUTOSAR. An ECU is in future AUTOSAR systems only one piece of the whole platform which host application software.

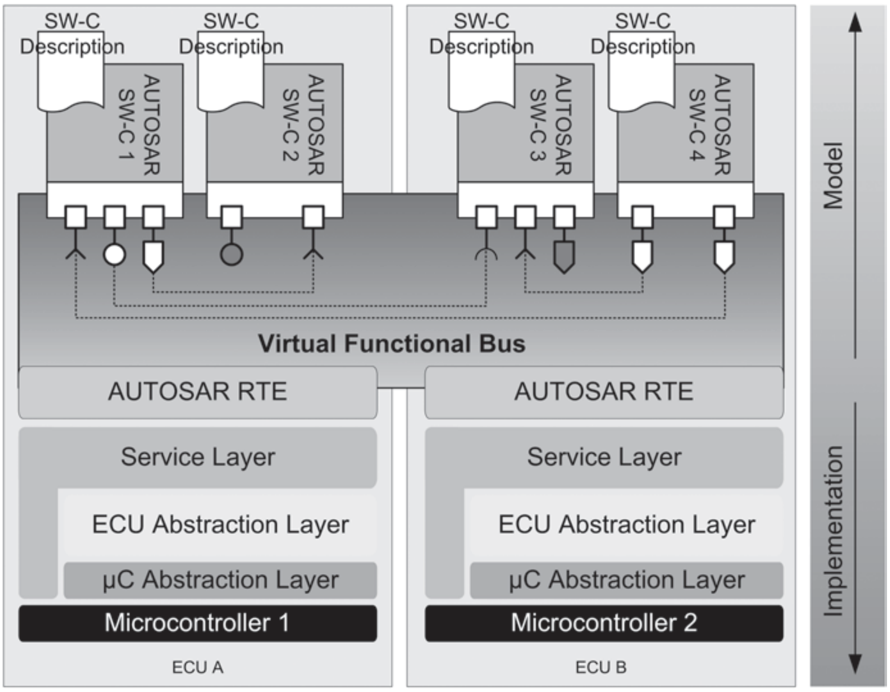


Fig. 4.17 From model to implementation. (Elektrobit Automotive GmbH, AUTOSAR—RTE User's Guide)

The requirements of the software components to their host environment are described in a standardized formal format. This format, the software component description, contains all information about the internal structure of the software component: Communication ports are described with their interfaces as well as the internal implementation details of the runtime parts. These runtime parts are called runnable entities and part of the description is how they are executed respectively by the event that triggers them. Trigger events can be set, e.g. by a cyclic clock or by the reception of a signal (data-received event).

The goal of the RTE is to hide all lower software layers. As such RTE hides the I/O driver layer and the communication layers as well as the access to the operating system. The application developer shall not write source code with direct access to counter, alarms and semaphores anymore. It is no longer necessary to define tasks and control respectively configure the task activation in the OS configuration. Instead, all runtime components of the applications will be represented by the runnable entities. It is the job of the RTE to manage all runnable entities, to map them into OS tasks and hence to predefine parts of the OS configuration. With this abstraction, it is possible to design, develop and test application software components

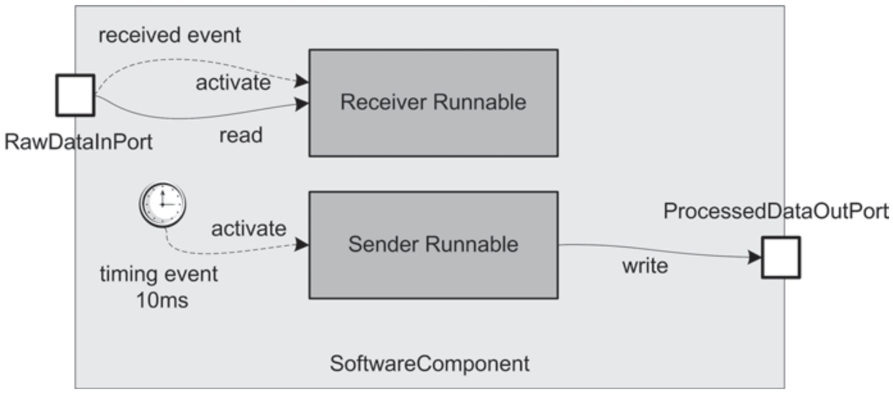


Fig. 4.18 Principle of a software component with ports and runnable entities (Elektrobit Automotive GmbH, AUTOSAR—RTE User's Guide)

independent of the final architecture. The configuration of the RTE, with all the architecture information, will be done later by the integration engineers (Fig. 4.18).

4.2.3 AUTOSAR Communication—An Example

4.2.3.1 Overview

The communication within an AUTOSAR architecture is running only across the ports of the application software components. The characteristic of a port is defined by its interface description. Available communication models are sender/receiver model for the transport of signals and client/server model for the execution of (remote) function calls. The receiver respectively the server can be on the same ECU (the so-called intra-ECU communication) or on different ECUs (the so-called inter-ECU communication). If the signals are shared across ECUs the data will be transported by the known automotive bus systems CAN, LIN and FlexRay.

With the abstraction of the communication channels, the bus protocols become invisible for the application developers. These developers do not need to know the details of the protocols or the register structure of the communication hardware module. The CAN module will be controlled by the CAN driver, and the "packaging" of the signals into CAN messages will be done by the COM module respectively by the CAN interface. These standard modules will be partly configured by the system configuration—there are communication parameters which have global validity. The CAN identifier and the position of signals in CAN frames (FlexRay, etc.) for all ECUs are the same and hence part of the system configuration. The message buffer configuration of the CAN hardware (e.g. used in first in, first out (FIFO) mode, ringbuffer or other) is part of the ECU configuration and will be defined by an integration expert who is integrating basic software with the application software.

4.2.3.2 Communication Path Through the Basic Software

AUTOSAR defines several modules which are taking part to send or receive a signal. On a first view, the modularity is complex and oversized. The advantages of the clear separation of functionality seem to be bought very costly—if at each module border, e.g. a buffer is needed to store the data temporarily. If each module is developed to be 100 % exchangeable the buffers may be needed, and if the modules are bundled it is possible to apply optimizations across the borders with the result that resources are not wasted. This can be done also if the configuration tooling has a clear view to the overall configuration, knowing all the facts and hence be able to provide an optimized configuration.

The following example shows how a signal will run through the AUTOSAR-layered architecture from a sender to a receiver application software component. The example refers to the CAN stack.

4.2.3.3 AUTOSAR Application

Within the application, information will be represented as a signal. For example, the actual vehicle speed could be an 8-bit value from 0 to 255, representing the speed in km/h. The port of the application, through which the signal is sent, is defined by an interface containing an 8-bit data element. The interface and the data type are defined within the software component description (XML). The generation of the RTE will also generate a component header (C-code) where these data types are defined. The RTE has two modes for the generation: In the contract phase, all c-headers with all data definitions are generated which are needed by the application software components (data types, constants, interface structures, function prototypes of the RTE API, etc.). In the second phase, the RTE generation phase, also the implementation of the functions will be generated. This is only possible if the complete path of the signal is known, i.e. the receiver and the medium for transport are defined (another software component on the same ECU, an output port of the microcontroller or something on another ECU).

If the interface and data types are defined in the software configuration description, the port can be configured. The name of the port is later a part of the RTE-send-API within the application software component. A runnable entity with the name *MyRunnable* can call the send function of the port *pPortname* with the value *Value* and data element *Speed* like shown in the following code-fragment:

```
void MyRunnable (Rte_Instance self)
{
 static SpeedValueType Value=0;
/* Compute Value */
…
/* Send Value */
 Rte_Send_pPortname_Speed(self, Value)
}
```
[4]

The parameter *self* is the instance handle of the runnable entity which is calling this function. This is needed to implement a methodology to use multiple instantiation of one (code) identical application software component. It is similar to the "this" pointer of a C++ class.

Calling the RTE send function will forward the value—depending on the configuration—to the lower software layers of the AUTOSAR architecture.

4.2.3.4 AUTOSAR RTE

The RTE contract phase generates prototypes of the send function. The RTE generation phase will generate the content of the send function. The RTE generator knows now the path of the signal. In this example, we assume that the signal shall be sent to another software component on another ECU by using the CAN bus. Hence, the RTE will forward the signal to the AUTOSAR COM module, which is responsible for all external communication. At the "sender signal mapping" (part of the ECU configuration), the interface data element of the port is mapped to a signal of the COM module (in the example the signal with the name COM_SIGNAL_1). The names can be different, but the data types have to be identical.

```
/* Rte_Send API function for DemoComponent */
Rte_StatusType Rte_Send_pPortname_Speed (SpeedType value)
{
 Status=Com_SendSignal (COM_SIGNAL_1, (void*) &value);
/* error handling, etc. */
}
```

If the receiver of this Speed-Signal would be on the same ECU, the implementation of this function could write direct into the memory of the receiver component. If this signal would be sent to a physical output port of the microcontroller, the I/O hardware abstraction would be called. This makes visible how deep the implementation of the RTE depends on the configuration. The RTE is simply the C-code representation of the configuration and unique for each ECU. In this example, the path continues at the AUTOSAR COM module.

[4] All used code examples are taken from the EB tresos AutoCore—they are only fragments and not complete.

4.2.3.5 AUTOSAR COM

The COM module receives the signal from the RTE and executes the "packaging" of the signal into a PDU. A PDU bundles several signals and will be forwarded to the lower layers. In the opposite way, the COM receives PDUs, extracts the signals and triggers actions of the RTE (e.g. by calling COM-callback-functions which fire an RTE event). The COM module implements many additional functions like timeout monitoring, byte conversion, different notification mechanisms, different transfer mode and so on. The lower levels of the communication stack do not know "signals" any more—they only handle PDUs.

The C-code of the application function and the RTE was straightforward and simple. The *Com_SendSignal* function of the COM module is very complex and does not fit to one page of paper. The core functionality of this function is the correct packaging of the signals into the PUD and the final triggering of the send mechanism of lower layers. Depending on the signal type, the packaging is complex or simple. A bool signal is easy; the correct placement of a byte array is more complex. For all operations, it is important to keep the data consistency, and the access shall be atomic. Hardware-specific library functions provide this atomic operation by using microcontroller-specific atomic functions.

If the signals are placed at the correct position of the corresponding PDU, two scenarios exist: If the PDU is send cyclic, it is sufficient to update the data—the function returns without further actions. In another context of the COM module all cyclic send operations will be processed independent, and the PDU will be sent to the configured time slot respectively with the configured cycle time. If the user defined, within the configuration, an instant sending of the signal or PDU, the lower layers will be called direct. From the COM module the path continues to the PDU router; the *PduId* for the message to be sent and a pointer to a complex data structure will be given to the transmit function:

```
PDU RouterPduR_ComTransmit(PduId, &pduinfo);
```

4.2.3.6 PDU Router

The job of the PDU router is to abstract the physical communication bus systems from the higher software layers. By using static routing tables the PDU router transfers PDUs from higher layers (e.g. the COM module) to lower layers (e.g. the CAN interface) and vice versa. The static routing tables, e.g. for the Tx-PDUs, contain for each PDU a target send function of the interface layer.

This is the first point in time where the concrete physical bus is relevant. The example signal shall be sent via the CAN bus; hence, the PDU will be handed over to the CAN interface. Defined by the configuration there exists a routing table entry for this PDU with a send function pointer to the CAN interface function *CanIf_Transmit*. The function *PduR_ComTransmit* itself is small and simple; it collects some information out of the configuration data structures and forwards all to the correct interface function. In this example, it is called the *CanIf_Transmit function*.

```
CanIf_Transmit(CanTxPduId, &pPduInfoPtr);
```

4.2.3.7 CAN Interface

The CAN interface is the last hardware-independent module in the AUTOSAR-layered architecture; it has the job to connect the hardware-dependent CAN driver. The CAN interface gets the PDUs from the above layers, adds the data the CAN driver needs and implements buffering functions (if the hardware is occupied) and acknowledgement functions for the send and receive signalling. Furthermore, the CAN interface controls the operating modes of the CAN controller (sleep mode and error modes) and signals changes of these modes to the higher layers (wake-up and bus-off).

The function *CanIf_Transmit*, which is called by the PDU router, takes the corresponding configuration for the given *PduId* and calls the CAN driver. The minimal configuration-related information is, for example, on which CAN channel the message shall be sent and which CAN ID shall be used. After assembling these data the driver function is called. It is possible that the CAN interface is able to call several drivers—e.g. if an external CAN module is used. In general, the on-chip CAN modules will be supported by one single CAN driver; hence, the use case with several CAN drivers is rare. The CAN driver function gets as parameter the unique hardware transmit handle (*Hth*) and a pointer to the data structure *PduInfo*, where the CAN ID, DLC and a pointer to the payload data are stored.

```
Can_Write(CanIf_CTxPdus[txPduIndex].hth, &canPduInfo);
```

4.2.3.8 CAN Driver

The CAN driver is the last piece of the communication path of an AUTOSAR compliant software platform. The driver initializes the register of the CAN module with correct values and sets the right bits for actions or data transfer. The access to these control and data registers will be provided by standardized functions to the higher software layers. Another important functionality is the implementation of all interrupt service routines Receive (Rx), Transmit (Tx), error, etc.). The driver receives the data to be transmitted by the CAN interface, copies it into a hardware-mailbox of the CAN controller and triggers the sending process. In the opposite direction, the CAN driver signals the reception of a new CAN message via callback mechanism, e.g. triggered by the receive interrupt.

Today's CAN controller provides complex, intelligent mechanisms implemented in hardware, for example, FIFO buffering or filtering mechanisms. Universal drivers, easy to port and maintainable on various hardware platforms, use these hardware functions not very often. For example, FIFO buffering implemented in software is not efficient; respectively, the valuable resource *CPU-time* of the microcontroller is dissipated in this case. With AUTOSAR, the hardware will become more comparable: With standardized interfaces, it is possible to do a benchmark test easier on the driver layer than on the direct hardware layer. With this comparability on driver layer, it becomes more important to have a highly optimized driver, which uses the available CAN controller hardware mechanisms perfect to save other resources. This is also valid for other protocol drivers, like the FlexRay driver, and of course for the peripheral drivers at all.

4.2.3.9 Conclusion

The AUTOSAR standard enables the possibility of microcontroller manufacturers to provide highly optimized driver for their hardware. These drivers can be used by the software vendors within their basic software platforms. With an overall accepted AUTOSAR standard, the huge investment into the drivers and higher software layers is safe—there is a chance to get a return on the investment. The CAN driver is now a piece of a much bigger picture and standalone nothing special, but together with the overall AUTOSAR concept, the model of a layered architecture makes sense. A legacy software was able to access direct the lowest software layer—the drivers. It prepares the data, implements all needed mechanisms and calls, e.g. *Can_Write* direct. This direct access was implemented in the past ECU development very often. Some developers already use quasi-standards for the driver layer to avoid double development of the same functionality. Within the higher software layers some modules were standardized, e.g. the OS was OSEK/VDX compliant. In future, it is needed to get a high total reuse rate for each single piece of software. A standardized platform like the described AUTOSAR platform gets more and more important for efficient software development. The use of this platform pays off as soon as there are first applications which can be completely reused for a second vehicle generation—not to be developed again due to a "simple" hardware change (due to cheaper hardware availability). Another contribution for cost reduction will be the reduction of integration effort for each single project: Due to standardized methods and increasing AUTOSAR knowledge in the overall development community, the integration will be faster. This cost reduction will only take effect if the AUTOSAR standard is used by many and every Tier 1 can use the same basic software platform for various OEMs. The success or failure of this standardization of automotive software architecture is in the hand of the AUTOSAR core members, especially the OEM. Only if all OEMs use the AUTOSAR standard "as it is", the overall AUTOSAR project will be successful.

4.3 Automotive Diagnostic Implementations on CAN

ISO 15765 Road vehicles—Diagnostic communication over Controller Area Network (DoCAN) was initially developed in the late 1990s. The purpose of the document set is to standardize the diagnostic communication-related OSI layers for the purpose of diagnostic and normal message communication.

Table 4.11 Enhanced and legislated OBD diagnostic specifications applicable to the OSI layers

Applicability	**OSI layers**	**Vehicle manufacturer enhanced diagnostics**	**Legislated OBD (On-Board Diagnostics)**		**Legislated WWH-OBD (World Wide Harmonized On-Board Diagnostics)**		
Seven layer according to ISO 7498-1 and ISO/IEC 10731	Application (layer 7)	ISO 14229-1 ISO 14229-3	ISO 15031-5		ISO 27145-3 ISO°14229-1		SAE J193 9, -71, -73
	Presentation (layer 6)	Vehicle Manufacturer specific or ISO 22901 ODX	ISO 15031-2, -5, -6, SAE°J1930-DA, SAE°J1979-DA, SAE°J2012-DA		ISO 27145-2, SAE °1930-DA, SAE°J1979-DA, SAE°J2012-DA, SAE°J1939 Top Level		SAE J193 9 Top Level
	Session (layer 5)	ISO 14229-2					N/A
	Transport (layer 4)	ISO 15765-2	ISO° 15765-2	ISO 15765-4	ISO 15765-4 / ISO°15765-2	ISO° 27145-4	SAE J193 9-21
	Network (layer 3)						SAE J193 9-31
	Data link (layer 2)	ISO 11898-1	ISO° 11898-1, -2		ISO 15765-4 / ISO°11898-1, -2		SAE J193 9-21
	Physical (layer 1)	ISO°11898-2, -3, -4, -5					SAE J193 9-15

ISO 15765 consists of the following parts, under the general title Road vehicles—DoCAN:

- Part 1: General information and use case definition.
- Part 2: Transport protocol and network layer services.
- Part 3: UDS on CAN implementation (UDSonCAN; will be replaced by ISO 14229-3 in 2010).
- Part 4: Requirements for emission-related systems.

Table 4.11 shows the relevant ISO and SAE standards as they are applicable to the OSI layers and how they relate to the following categories (columns in Table 4.1):

- Vehicle manufacturer-enhanced diagnostics.
- Legislated On-Board Diagnostics (OBD).
- Legislated World-Wide Harmonized On-Board Diagnostics (WWH-OBD): shows the legislator-approved ISO and Society of Automotive Engineers (SAE) implementations.

The application layer (OSI 7) services for the:

- Vehicle manufacturer-enhanced diagnostics are defined in ISO 14229-3 UDSonCAN (former ISO 15765-3). This part of ISO 14229-1 is an implementation of Part 1 onto CAN.
- Legislated OBD are defined in ISO 15031-5 Road vehicles—Communication between vehicle and external equipment for emission-related diagnostics—Part 5: emission-related diagnostic services. This standard is referenced by the European Commission (EC) directives for EURO 4, 5, 6 (passenger cars and

light-duty vehicles (LDV)) and directives for EURO IV, V, VI (heavy-duty vehicles (HDVs)). The technical equivalent standard of ISO 15031-5 is SAE 1979 E/E diagnostic test modes, which is referenced by the California Air Resources Board (ARB) and Environmental Protection Agency (EPA) for the federal states.

- Legislated WWH-OBD is defined in ISO 27145 Road vehicles—Implementation of emission-related WWH-OBD communication requirements. This upcoming standard is still under development (2010) and will long term replace the ISO 15031-5/SAE J1979 standards. The legislators in Europe and the USA have also approved several parts of SAE J1939 Recommended Practice for a Serial Control and Communications Vehicle Network:
  - SAE 1939-73, Application Layer—Diagnostics (includes DM—diagnostic modes and data items: parameter group numbers (PGNs) and suspect parameter numbers (SPNs).
  - SAE 1939-71, Vehicle Application Layer (includes normal in-vehicle communication messages and data items: SPNs).

The presentation layer (OSI 6) services for the:

- Vehicle manufacturer-enhanced diagnostics are:
  - either specific to the manufacturer or
  - the manufacturer defines the diagnostic data according to ISO 22901, Road vehicles—Open Diagnostic data eXchange (ODX).
- Legislated OBD are defined in:
  - ISO 15031-2, Road vehicles—Communication between vehicle and external equipment for emission-related diagnostics—Part 2: Guidance on terms, definitions, abbreviations and acronyms with reference to the Digital Annex of SAE°J1930-DA, electrical/electronic systems diagnostic terms, definitions, abbreviations and acronyms (includes all standardized terms and abbreviations).
  - ISO 15031-5, Road vehicles—Communication between vehicle and external equipment for emission-related diagnostics—Part 5: Emission-related diagnostic services with reference to the Digital Annex of SAE°J1979-DA E/E diagnostic test modes (includes all standardized data items: parameter identifiers (PIDs), test identifiers (TIDs), monitor identifiers (MIDs) and infoType identifiers (ITIDs).
  - ISO 15031-6, Road vehicles—Communication between vehicle and external equipment for emission-related diagnostics—Part 6: with reference to the Digital Annex of SAE°J2012-DA, diagnostic trouble codes (includes all standardized DTCs).
- Legislated WWH-OBD are defined in:
  - ISO 27145-2 Road vehicles—Implementation of WWH-OBD communication requirements—Part 2: Common data dictionary (CDD). This standard references the same SAE Digital Annexes as ISO 15031 (emission-related OBD):
    - SAE°J1930-DA, electrical/electronic systems diagnostic terms, definitions, abbreviations and acronyms (includes all standardized terms and abbreviations).

  - SAE°J1979-DA, E/E diagnostic test modes (includes all standardized data items: PIDs, TIDs, MIDs, ITIDs).
  - SAE°J2012-DA, diagnostic trouble codes (includes all standardized DTCs).
  - SAE°J1939, top level (includes all standardized SPNs and PGNs).

The session layer (OSI 5) services are defined in ISO 14229-2 Road vehicles—Unified diagnostic services (UDSs)—Part 2: Session layer services. This standard provides a protocol-independent standardized service primitive interface between application layer services and transport protocol/network layer services. This standard is not limited to the standards as listed in Table 4.11.

The transport protocol layer (OSI 4) and network layer (OSI 3) services are defined in:

- ISO 15765-2 Road vehicles—DoCAN—Part 2: Transport protocol and network layer services. This part of the standard is applicable to all three categories. The legislated categories (OBD and WWH-OBD) have additional requirements defined in ISO 15765-4, which are related to CAN identifier definition, transport protocol timing, etc.
- SAE 1939-21, transport protocol is only applicable to WWH-OBD.
- SAE 1939-31, network layer is only applicable to WWH-OBD.

The data link layer (OSI 2) requirements are defined in:

- ISO 11898-1, Road vehicles—Controller area network (CAN)—Part 1: Data link layer and physical signalling. This standard is applicable to all CAN networks independent of the upper OSI layer implementations.
- SAE 1939-21, Data link layer is only applicable to WWH-OBD.

The physical layer (OSI 1) requirements are defined in various parts of ISO 11898.

- Vehicle manufacturer-enhanced diagnostics can be implemented on any of the following physical layer standards:
  - ISO°11898-2, Road vehicles—Controller area network (CAN)—Part 2: High-speed medium access unit (majority of automotive implementations).
  - ISO°11898-3, Road vehicles—Controller area network (CAN)—Part 3: Low-speed, fault-tolerant, medium-dependent interface.
  - ISO°11898-4, Road vehicles—Controller area network (CAN)—Part 4: Time-triggered communication.
- Legislated OBD is only allowed on ISO°11898-2, Road vehicles—Controller area network (CAN)—Part 2: High-speed medium access unit.
- Legislated WWH-OBD is allowed on:
  - ISO°11898-2, Road vehicles—Controller area network (CAN)—Part 2: High-speed medium access unit.
  - SAE J1939-15, Reduced Physical Layer, 250K bits/sec, Un-Shielded Twisted Pair (UTP).

Figure 4.19 illustrates the most applicable application implementations utilizing the DoCAN protocol.

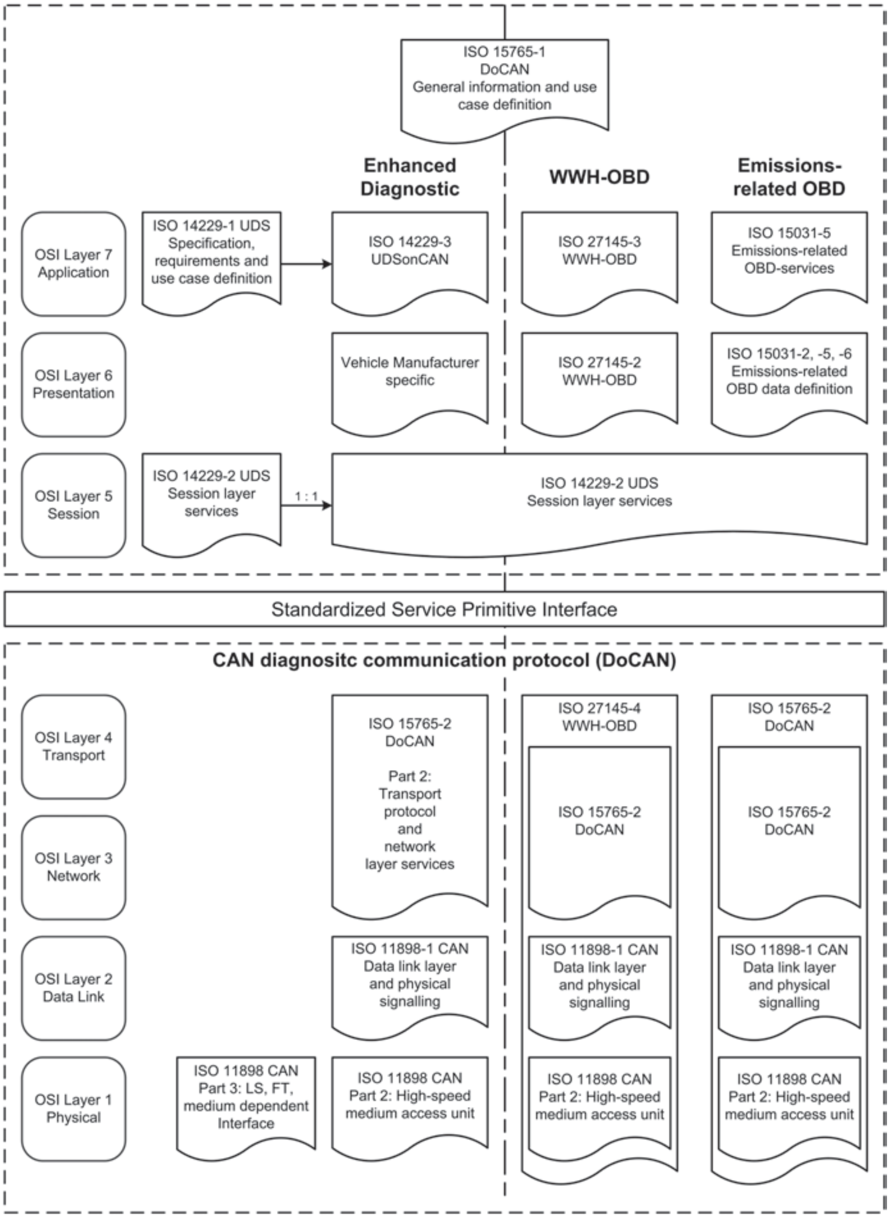

ISO 15765-1 DoCAN General information and use case definition
Enhanced Diagnostic
WWH-OBD
Emissions-related OBD
OSI Layer 7 Application
ISO 14229-1 UDS Specification, requirements and use case definition
ISO 14229-3 UDSonCAN
ISO 27145-3 WWH-OBD
ISO 15031-5 Emissions-related OBD-services
OSI Layer 6 Presentation
Vehicle Manufacturer specific
ISO 27145-2 WWH-OBD
ISO 15031-2, -5, -6 Emissions-related OBD data definition
OSI Layer 5 Session
ISO 14229-2 UDS Session layer services
1 : 1
ISO 14229-2 UDS Session layer services
Standardized Service Primitive Interface
CAN diagnositc communication protocol (DoCAN)
OSI Layer 4 Transport
OSI Layer 3 Network
ISO 15765-2 DoCAN Part 2: Transport protocol and network layer services
ISO 27145-4 WWH-OBD
ISO 15765-2 DoCAN
ISO 15765-2 DoCAN
ISO 15765-2 DoCAN
OSI Layer 2 Data Link
ISO 11898-1 CAN Data link layer and physical signalling
OSI Layer 1 Physical
ISO 11898 CAN Part 3: LS, FT, medium dependent Interface
ISO 11898 CAN Part 2: High-speed medium access unit

Fig. 4.19 Diagnostic communication over CAN document reference according to OSI model

4.3.1 OBDonCAN—ISO 15031 emissions-related OBD on ISO 15765-4 DoCAN

With the introduction of ISO 15765 DoCAN as a legislator-approved emission-related system OBD data link the ISO 15031-5/SAE J1979 emission-related diagnostic services/test modes have been slightly modified to benefit from the ISO 15765-2 DoCAN transport protocol and network layer services standard. ECU response messages, which used to be split into multiple response message for the non-CAN protocols (SAE J1850, ISO 9141-2 and ISO 14230-4), are now assembled into one response message, consisting of multiple CAN frames. The payload data of the PDU are identical between all ISO 15031-5/SAE J1979-defined diagnostic messages (DMs).

The requirements and features of the OBDonCAN protocol can be summarized as follows:

- Tester can request up to sic PIDs in a single functionally addressed request message. Non-CAN protocols allow only for one PID in the request message.
- Either baud rate is allowed:
  - 250 kBit/s
  - 500 kBit/s
- A maximum of eight emission-related OBD ECUs is allowed to respond on ISO 15031-5 requests.
- Tester is only allowed to send functionally addressed request messages (11 bit CAN ID = 0 × 7DF). Flow control from the tester and response messages from the ECU use the physical request and response CAN IDs (Table 4.12).
- Vehicle manufacturer, which decides to use 29-bit CAN IDs on their CAN bus, must use the 29-bit CAN IDs as specified in ISO 15765-4 (Table 4.13).
- The ISO 15765-2 DoCAN transport protocol and network layer services standard supports up to 4.095 bytes in a message. Reception of messages up to six (6) or seven (7) data bytes is performed via reception of a unique N_PDU (Figs. 4.20 and 4.21).
- The ISO 15765-2 DoCAN multiple-frame message transfer. The flow control mechanism allows the receiver to inform the sender about the receiver's capabilities. Since different nodes may have different capabilities, the flow control sent by the receiver informs the sender about its capabilities. The sender shall conform to the receiver's capabilities (Fig. 4.22).
- The ISO 15765-2 DoCAN—Transport protocol and network layer services timeout and performance requirements are more stringent in ISO 15765-4 (Table 4.14).
- The external test equipment shall use the following network layer parameter values for its FlowControl frames, sent in response to the reception of a FirstFrame (Table 4.15 and 4.16).

Table 4.12 11-bit legislated OBD/WWH-OBD CAN identifiers

CAN identifier	Description
0 × 7DF	CAN identifier for functionally addressed request messages sent by external test equipment
0 × 7E0	Physical request CAN identifier from external test equipment to ECU #1
0 × 7E8	Physical response CAN identifier from ECU #1 to external test equipment
0 × 7E1	Physical request CAN identifier from external test equipment to ECU #2
0 × 7E9	Physical response CAN identifier from ECU #2 to external test equipment
0 × 7E2	Physical request CAN identifier from external test equipment to ECU #3
0 × 7EA	Physical response CAN identifier from ECU #3 to external test equipment
0 × 7E3	Physical request CAN identifier from external test equipment to ECU #4
0 × 7EB	Physical response CAN identifier ECU #4 to the external test equipment
0 × 7E4	Physical request CAN identifier from external test equipment to ECU #5
0 × 7EC	Physical response CAN identifier from ECU #5 to external test equipment
0 × 7E5	Physical request CAN identifier from external test equipment to ECU #6
0 × 7ED	Physical response CAN identifier from ECU #6 to external test equipment
0 × 7E6	Physical request CAN identifier from external test equipment to ECU #7
0 × 7EE	Physical response CAN identifier from ECU #7 to external test equipment
0 × 7E7	Physical request CAN identifier from external test equipment to ECU #8
0 × 7EF	Physical response CAN identifier from ECU #8 to external test equipment

While not required for current implementations, it is strongly recommended (and may be required by applicable legislation) that for future implementations the following 11-bit CAN identifier assignments be used:
0 × 7E0/0 × 7E8 for engine control module (ECM)
0 × 7E1/0 × 7E9 for transmission control module (TCM)

Table 4.13 Summary of 29-bit CAN identifier format—normal fixed addressing

CAN-Id ID bit position	28	24	23	16	15	8	7	0
Functional CAN IDId	0 × 18		0xDB		TA		SA	
Physical CAN IDId	0 × 18		0xDA		TA		SA	

Note: The CAN identifier values given in this table use the default value for the priority information in accordance with ISO 15765-2

4.3.2 UDSonCAN—ISO 14229-3

As mentioned earlier, UDSonCAN is originally based on ISO 15765-3. The revision of ISO 15765-3 currently under work at ISO will be published as ISO 14229-3. The reason for the renumbering of the well-established standard in the automotive industry is based on the fact that ISO 14229-1 UDS shall be implemented on many different protocols and data links (CAN, FlexRay, Internet Protocol, K-Line, etc.) according to the OSI model. Fig. 4.23 illustrates the implementation concept of UDS.

ISO 14229-1 UDS specifies the DMs in a protocol-independent manner. This document is currently under revision by the ISO TC22/SC2/WG1/TF5 diagnostic requirements task force. All lessons learnt and implementation feedback from system suppliers, vehicle manufacturers and tool suppliers is under implementation.

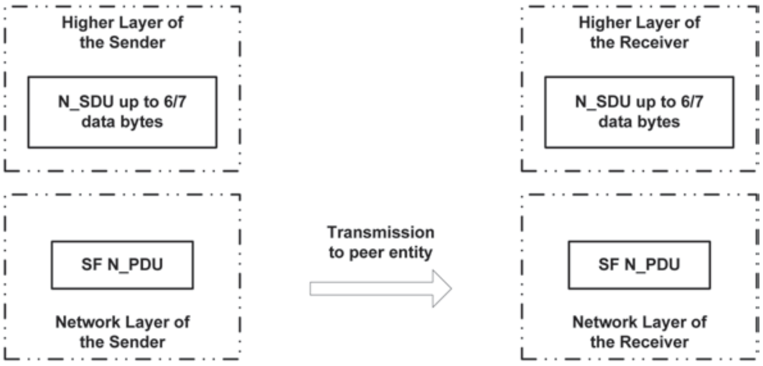

Higher Layer of the Sender
N_SDU up to 6/7 data bytes
SF N_PDU
Network Layer of the Sender
Transmission to peer entity
Higher Layer of the Receiver
N_SDU up to 6/7 data bytes
SF N_PDU
Network Layer of the Receiver

Fig. 4.20 Example of a single-frame (SF) transmission

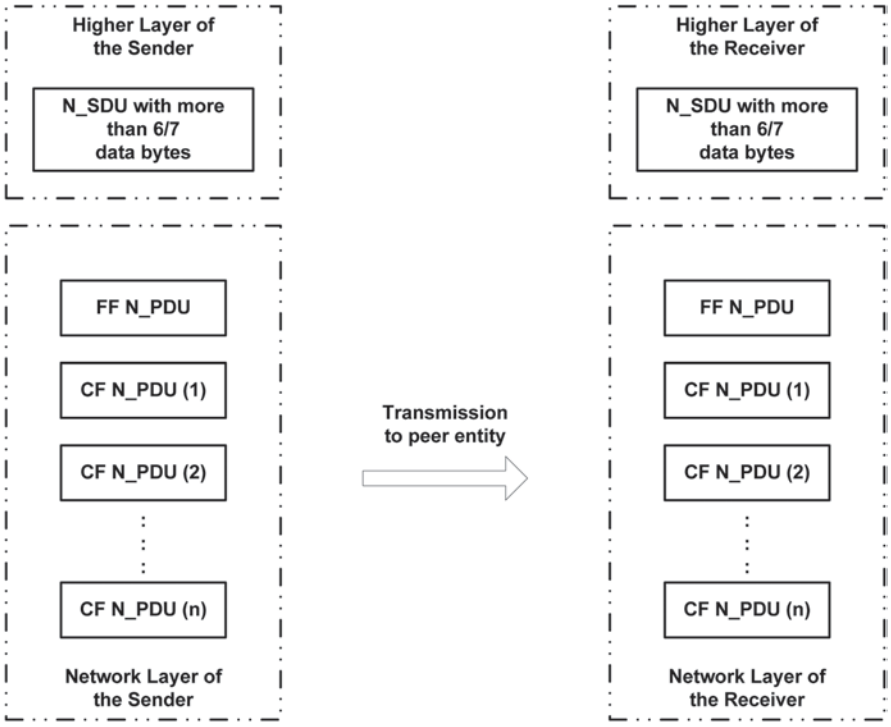

Higher Layer of the Sender
N_SDU with more than 6/7 data bytes
FF N_PDU
CF N_PDU (1)
CF N_PDU (2)
CF N_PDU (n)
Network Layer of the Sender
Transmission to peer entity
Higher Layer of the Receiver
N_SDU with more than 6/7 data bytes
FF N_PDU
CF N_PDU (1)
CF N_PDU (2)
CF N_PDU (n)
Network Layer of the Receiver

Fig. 4.21 Example of a multiple-frame transmission (segmentation and reassembly)

ISO 14229-2 UDS specifies common session layer services to provide independence between UDS (Part 1) and all network/transport protocols (ISO 15765-2 CAN, ISO/WD 10681-2 FlexRay, ISO/CD 13400-2 DoIP, ISO 14230-2 K-Line, LIN, MOST, etc.).

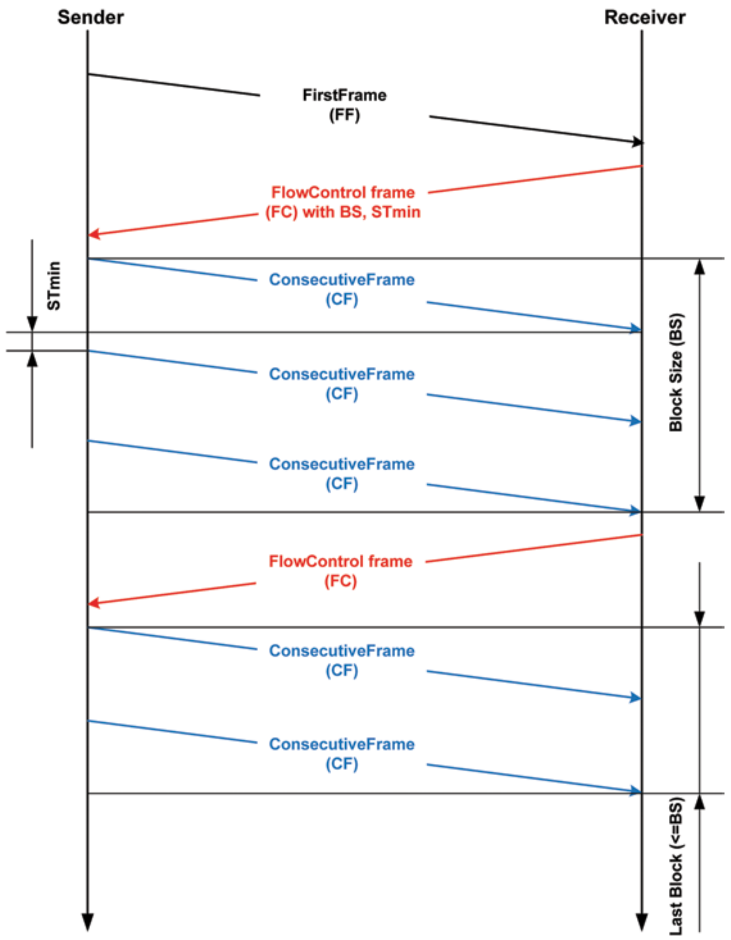


Fig. 4.22 Flow control (FC) mechanism

A common service primitive interface is specified between OSI layer 4 (Transport) and layer 5 (Session) via the so-called service request/confirmation/indication primitives. This interface allows seamless implementation of ISO°14229-1 UDSs with any communication protocol titled "DoXYZ/CoXYZ" like ISO°15765 DoCAN—diagnostic communication over controller area network, ISO°13400 DoIP—diagnostic communication over Internet protocol, ISO°10681 CoFlexRay—communication over FlexRay, ISO°14230 DoK-Line—diagnostic communication over K-line.

ISO 14229-3 UDSonCAN is based on a common document template for UDS implementations on a specific protocol like ISO 15765 DoCAN. This specific part of ISO 14229 specifies implementation requirements related to the following:

- The diagnostic services to be used for diagnostic communication over CAN.
- The server memory programming for all in-vehicle servers connected to a CAN network with an external test equipment.

Table 4.14 Network layer timeout and performance requirement values

Parameter	Timeout value	Performance requirement value
N_As/N_Ar	25 ms	–
N_Bs	75 ms	–
N_Br	–	(N_Br+N_Ar)<25 ms
N_Cs	–	(N_Cs+N_As)<50 ms
N_Cr	150 ms	–

Table 4.15 External test equipment network layer parameter values

Parameter	Name	Value	Description
N_WFTmax	WaitFrame transmission	0	No FlowControl wait frames are allowed for legislated OBD/WWH-OBD. The FlowControl frame sent by the external test equipment following the FirstFrame of an ECU response message shall contain the FlowStatus FS set to 0 (ClearToSend), which forces the ECU to start immediately after the reception of the FlowControl frame with the transmission of the ConsecutiveFrame(s)
BS	BlockSize	0	A single FlowControl frame shall be transmitted by the external test equipment for the duration of a segmented message transfer. This unique FlowControl frame shall follow the FirstFrame of an ECU response message
STmin	SeparationTime	0	This value allows the ECU to send ConsecutiveFrames, following the FlowControl frame sent by the external test equipment, as fast as possible

If a reduced implementation of the ISO°15765-2 network layer is done in a legislated OBD/WWH-OBD ECU, covering only the above-listed FlowControl frame parameter values (BS, STmin), then any FlowControl frame received during legislated OBD/WWH-OBD communication and using different FlowControl frame parameter values as defined in this table shall be ignored by the receiving legislated OBD/WWH-OBD ECU (treated as an unknown network layer PDU)

It does not contain information related to any requirement for the in-vehicle CAN bus architecture.

Figure 4.24 illustrates all referenced documents according to the OSI layers.

4.3.2.1 UDSonCAN Services Overview

The purpose of Table 4.17 is to reference all UDSs as they are applicable for an implementation of UDSonCAN. The table contains the sum of all applicable services. Certain applications using this part of ISO 14229 to implement UDSonCAN may restrict the number of useable services and may categorize them in certain application areas/diagnostic sessions (default session, programming session, etc.).

Table 4.16 Emission-related diagnostic service definition OBDonCAN (ISO 15031-5 on ISO 15765-4)

Service Id	Description
0 × 01	Request current powertrain diagnostic data The purpose of this service is to allow access to current emission-related data values, including analog inputs and outputs, digital inputs and outputs and system status information. The request for information includes a parameter-identification (PID) value that indicates to the on-board system the specific information requested. PID specifications, scaling information and display formats are included in SAE J1979-DA
0 × 02	Request powertrain freeze frame data The purpose of this service is to allow access to emission-related data values in a freeze frame. This allows expansion to meet manufacturer-specific requirements not necessarily related to the required freeze frame, and not necessarily containing the same data values as the required freeze frame. The request message includes a parameter identification (PID) value that indicates to the on-board system the specific information requested. PID specifications, scaling information and display formats for the freeze frame are included in SAE J1979-DA
0 × 03	Request emission-related diagnostic trouble codes The purpose of this service is to enable the external test equipment to obtain "confirmed" emission-related DTCs Send a Service 0 × 03 request for all emission-related DTCs. Each ECU that has DTCs shall respond with one (1) message containing all emission-related DTCs. If an ECU does not have emission-related DTCs, then it shall respond with a message indicating no DTCs are stored by setting the parameter of DTC to 0 × 00
0 × 04	Clear/reset emission-related diagnostic information The purpose of this service is to provide a means for the external test equipment to command ECUs to clear all emission-related diagnostic information. This includes the following: • MIL and number of diagnostic trouble codes (can be read with Service 0 × 01, PID 0 × 01) • Clear the inspection/maintenance (I/M) readiness bits (can be read with Service 0 × 01, PID 0 × 01) • Confirmed diagnostic trouble codes (can be read with Service 0 × 03) • Pending diagnostic trouble codes (can be read with Service 0 × 07) • Diagnostic trouble code for freeze frame data (can be read with Service 0 × 02, PID 0 × 02) • Freeze frame data (can be read with Service 0 × 02) • Status of system monitoring tests (can be read with Service 0 × 01, PID 0 × 41 • On-board monitoring test results (can be read with Service 0 × 06) • Distance travelled while MIL is activated (can be read with Service 0 × 01, PID 0 × 21) • Number of warm-ups since DTCs cleared (can be read with Service 0 × 01, PID 0 × 30) • Distance travelled since DTCs cleared (can be read with Service 0 × 01, PID 0 × 31) • Engine run time while MIL is activated (can be read with Service 0 × 01, PID 0 × 4D) • Engine run time since DTCs cleared (can be read with Service 0 × 01, PID 0 × 4E) • Reset misfire counts of standardized test ID 0 × 0B to zero (can be read with Service 0 × 06)

Table 4.16 (continued)

Service Id	Description
0 × 05	Request oxygen sensor monitoring test results Service 0 × 05 is not supported for ISO 15765-4. The functionality of Service 0 × 05 is implemented in Service 0 × 06
0 × 06	Request on-board monitoring test results for specific monitored systems The purpose of this service is to allow access to the results for on-board diagnostic monitoring tests of specific components/systems that are continuously monitored (e.g. misfire monitoring for gasoline vehicles) and non-continuously monitored (e.g. catalyst system)
0 × 07	Request emission-related diagnostic trouble codes detected during current or last completed driving cycle The purpose of this service is to enable the external test equipment to obtain "pending" diagnostic trouble codes detected during current or last completed driving cycle for emission-related components/systems. Service 0 × 07 is required for all DTCs and is independent of Service 0 × 03. The intended use of this data is to assist the service technician after a vehicle repair, and after clearing diagnostic information, by reporting test results after a single driving cycle. If the test failed during the driving cycle, the DTC associated with that test shall be reported. Test results reported by this service do not necessarily indicate a faulty component/system. If test results indicate a failure after additional driving, then the MIL will be illuminated and a DTC will be set and reported with Service 0 × 03, indicating a faulty component/system. This service can always be used to request the results of the latest test, independent of the setting of a DTC
0 × 08	Request control of on-board system, test or component The purpose of this service is to enable the external test equipment to control the operation of an on-board system, test or component The data bytes will be specified, if necessary, for each test ID in SAE J1979-DA, and will be unique for each test ID Possible uses for these data bytes in the request message are as follows: • Turn on-board system/test/component ON • Turn on-board system/test/component OFF • Cycle on-board system/test/component for 'n' seconds Possible uses for these data bytes in the response message are as follows: • Report system status • Report test results
0 × 09	Request vehicle information The purpose of this service is to enable the external test equipment to request vehicle-specific vehicle information such as vehicle identification number (VIN) and calibration IDs. Some of this information may be required by regulations and some may be desirable to be reported in a standard format if supported by the vehicle manufacturer. InfoTypes are defined in SAE J1979-DA
0 × 0A	Request emission-related diagnostic trouble codes with permanent status The purpose of this service is to enable the external test equipment to obtain all DTCs with "permanent DTC" status. These are DTCs that are "confirmed" and are retained in the non-volatile memory of the server until the appropriate monitor for each DTC has determined that the malfunction is no longer present and is not commanding the MIL on Service 0 × 0A is required for all emission-related DTCs. The intended use of this data is to prevent vehicles from passing an in-use inspection simply by disconnecting the battery or clearing DTCs with a scan tool prior to the inspection. The presence of permanent DTCs at an inspection without the MIL illuminated is an indication that a proper repair was not verified by the on-board monitoring system

Table 4.16 (continued)

Service Id	Description
	Permanent DTCs shall be stored in non-volatile memory (NVRAM) and may not be erased by any diagnostic services (generic or enhanced) or by disconnecting power to the ECU A confirmed DTC shall be stored as a permanent DTC no later than the end of the ignition cycle and subsequently at all times that the confirmed DTC is commanding the MIL on (e.g. for currently failing systems but not during the 40 warm-up cycle self-healing process) Permanent DTCs may be erased if: • The OBD system itself determines that the malfunction that caused the permanent fault code to be stored is no longer present and is not commanding the MIL on, e.g. three consecutive complete driving cycles with no malfunction, or as specified by the OBD regulations • After clearing fault information in the ECU (i.e. through the use of a diagnostic service or battery disconnect): • For monitors subject to minimum in-use ratio requirement, the diagnostic monitor for the malfunction that caused the permanent DTC to be stored has fully executed (i.e. has executed the minimum number of checks necessary for MIL illumination) and determined the malfunction is no longer present, e.g. one complete driving cycle with no malfunction or as specified by the OBD regulations • For monitors not subject to minimum in-use ratio requirement, the diagnostic monitor for the malfunction that caused the permanent DTC to be stored has fully executed (i.e. has executed the minimum number of checks necessary for MIL illumination) and determined the malfunction is no longer present, e.g. one complete driving cycle with no malfunction or as specified by the OBD regulations and the vehicle has completed a standard driving cycle used to increment the in-use general denominator • Permanent fault codes may be erased when the ECU containing the permanent DTCs is reprogrammed if the readiness status for all monitored components and systems is set to "not complete" in conjunction with the reprogramming event

4.3.3 Development Trends

4.3.3.1 History

Today two CAN-based protocols exist which are legislator-approved protocols for emission-related OBD:

- ISO 15031-5 on ISO 15765-4 OBDonCAN
- SAE J1939

ISO 15031-5 is technically identical with SAE J1979 and defines specific services/test modes to exchange data with the emission-related system in the vehicle. The services support the request of current PID information, stored freeze frame data, readout of DTC: current, pending and permanent status, clearing of DTC-related data, readout of monitoring test results, request control of on-board system (test or component) and request of vehicle information.

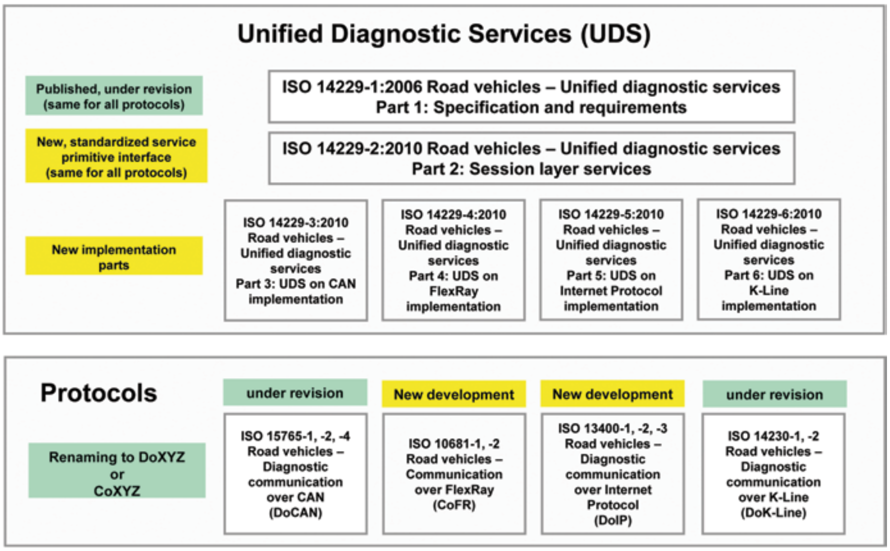

Unified Diagnostic Services (UDS)
Published, under revision (same for all protocols)
New, standardized service primitive interface (same for all protocols)
New implementation parts
ISO 14229-1:2006 Road vehicles – Unified diagnostic services Part 1: Specification and requirements
ISO 14229-2:2010 Road vehicles – Unified diagnostic services Part 2: Session layer services
ISO 14229-3:2010 Road vehicles – Unified diagnostic services Part 3: UDS on CAN implementation
ISO 14229-4:2010 Road vehicles – Unified diagnostic services Part 4: UDS on FlexRay implementation
ISO 14229-5:2010 Road vehicles – Unified diagnostic services Part 5: UDS on Internet Protocol implementation
ISO 14229-6:2010 Road vehicles – Unified diagnostic services Part 6: UDS on K-Line implementation
Protocols
Renaming to DoXYZ or CoXYZ
under revision
New development
New development
under revision
ISO 15765-1, -2, -4 Road vehicles – Diagnostic communication over CAN (DoCAN)
ISO 10681-1, -2 Road vehicles – Communication over FlexRay (CoFR)
ISO 13400-1, -2, -3 Road vehicles – Diagnostic communication over Internet Protocol (DoIP)
ISO 14230-1, -2 Road vehicles – Diagnostic communication over K-Line (DoK-Line)

Fig. 4.23 Implementation concept of unified diagnostic services (UDSs)

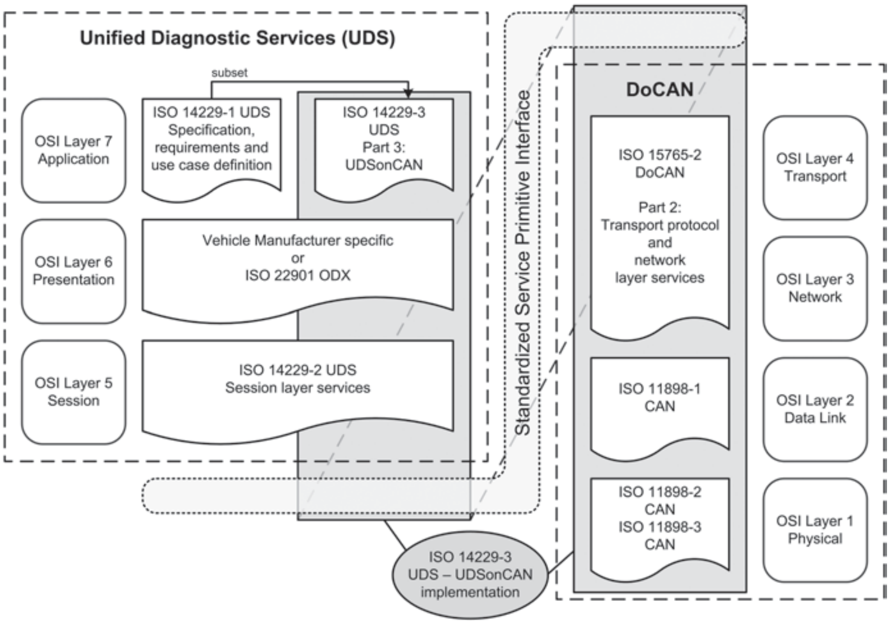

Unified Diagnostic Services (UDS)
subset
OSI Layer 7 Application
ISO 14229-1 UDS Specification, requirements and use case definition
ISO 14229-3 UDS Part 3: UDSonCAN
OSI Layer 6 Presentation
Vehicle Manufacturer specific or ISO 22901 ODX
OSI Layer 5 Session
ISO 14229-2 UDS Session layer services
Standardized Service Primitive Interface
DoCAN
ISO 15765-2 DoCAN Part 2: Transport protocol and network layer services
ISO 11898-1 CAN
ISO 11898-2 CAN ISO 11898-3 CAN
OSI Layer 4 Transport
OSI Layer 3 Network
OSI Layer 2 Data Link
OSI Layer 1 Physical
ISO 14229-3 UDS – UDSonCAN implementation

Fig. 4.24 Overview of unified diagnostic services to the OSI layers

Table 4.17 Overview of applicable ISO 14229-1 unified diagnostic services and data ranges

Diagnostic service name (ISO 14229-1)	SID value	SFID value	Sub-function name
Diagnostic and communication management functional unit			
DiagnosticSessionControl	0 × 10	0 × 01	defaultSession
		0 × 02	programmingSession
		0 × 03	extendedSession
		0 × 04	safetySystemDiagnosticSession
ECUReset	0 × 11	0 × 01	hardReset
		0 × 02	keyOffOnReset
		0 × 03	softReset
		0 × 04	enableRapidPowerShutDown
		0 × 05	disableRapidPowerShutDown
SecurityAccess	0 × 27	0 × 01	requestSeed
		0 × 02	sendKey
CommunicationControl	0 × 28	0 × 00	enableRxAndTx
		0 × 01	enableRxAndDisableTx
		0 × 02	disableRxAndEnableTx
		0 × 03	disableRxAndTx
TesterPresent	0 × 3E	0 × 00	zeroSubFunction
SecuredDataTransmission	0 × 84	N/A	N/A
ControlDTCSetting	0 × 85	0 × 01	on
		0 × 02	off
ResponseOnEvent	0 × 86	0 × 00	stopResponseOnEvent
		0 × 01	onDTCStatusChange
		0 × 02	onTimerInterrupt
		0 × 03	onChangeOfDataIdentifier
		0 × 04	reportActivatedEvents
		0 × 05	startResponseOnEvent
		0 × 06	clearResponseOnEvent
		0 × 07	onComparisonOfValues
LinkControl	0 × 87	0 × 01	verifyModeTransitionWithFixed-Parameter
		0 × 02	verifyModeTransitionWith-Specific-Parameter
		0 × 03	transitionMode
Data Transmission Functional Unit			
ReadDataByIdentifier	0 × 22	–	N/A
ReadMemoryByAddress	0 × 23	–	N/A
ReadScalingDataByIdentifier	0 × 24	–	N/A
ReadDataByPeriodicIdentifier	0 × 2A	–	N/A
DynamicallyDefineData-Identifier	0 × 2C	0 × 01	defineByIdentifier
		0 × 02	defineByMemoryAddress
		0 × 03	clearDynamicallyDefinedDataIdenti-fier
WriteDataByIdentifier	0 × 2E	–	N/A
WriteMemoryByAddress	0 × 3D	–	N/A
Stored data transmission functional unit			
ReadDTCInformation	0 × 19	0 × 01	reportNumberOfDTCByStatusMask

Table 4.17 (continued)

Diagnostic service name (ISO 14229-1)	SID value	SFID value	Sub-function name
		0 × 02	reportDTCByStatusMask
		0 × 03	reportDTCSnapshotIdentification
		0 × 04	reportDTCSnapshotRecordByDTC-Number
		0 × 05	reportDTCSnapshotRecordByRecord-Number
		0 × 06	reportDTCExtendedDataRecordBy-DTCNumber
		0 × 07	reportNumberOfDTCBySeverityMas-kRecord
		0 × 08	reportDTCBySeverityMaskRecord
		0 × 09	reportSeverityInformationOfDTC
		0 × 0A	reportSupportedDTC
		0 × 0B	reportFirstTestFailedDTC
		0 × 0C	reportFirstConfirmedDTC
		0 × 0D	reportMostRecentTestFailedDTC
		0 × 0E	reportMostRecentConfirmedDTC
		0 × 0F	reportMirrorMemoryDTCByStatus-Mask
		0 × 10	reportMirrorMemoryDTCExtend-edDataRecordByDTCNumber
		0 × 11	reportNumberOfMirrorMemoryD-TCByStatusMask-
		0 × 12	reportNumberOfEmissions-Relate-dOBDDTCByStatusMask
		0 × 13	reportEmissionsRelatedOBD-DTCByStatusMask
		0 × 14	reportDTCFaultDetectionCounter
		0 × 15	reportDTCWithPermanentStatus
		0 × 41	reportWWHOBDNumberOfDTCBy-MaskRecord
		0 × 42	reportWWHOBDDTCByMaskRecord
		0 × 55	reportWWHOBDDTCWithPerma-nentStatus
ClearDiagnosticInformation	0 × 14	–	N/A
Input/output control functional unit			
InputOutputControlByIdentifier	0 × 2F	–	N/A
Remote activation of routine functional unit			
RoutineControl	0 × 31	0 × 01	startRoutine
		0 × 02	stopRoutine
		0 × 03	requestRoutineResults
Upload/download functional unit			
RequestDownload	0 × 34	–	N/A
RequestUpload	0 × 35	–	
TransferData	0 × 36	–	
RequestTransferExit	0 × 37	–	

The ISO 15031-5 diagnostic services are very specific to emission-related OBD systems. The automotive industry has developed the so-called enhanced diagnostic protocols like ISO 14230 Keyword Protocol 2000 and SAE J2190 E/E Enhanced diagnostic test modes to be able to diagnose non-emission-related OBD systems as well as the functionality beyond the requirements included in the legislation.

At the time when the emission-related systems of the HDVs were referenced in the legislation (with EURO IV), the SAE J1939 protocol became an allowed data link in addition to ISO 15031-5. Both protocols support the CAN bus but are very different in their messages and functionality.

Since that time the HDV manufacturers have two choices to fulfil the emission-related OBD regulations. Two different diagnostic tester implementations are required to diagnose the HDV's emission-related OBD systems.

4.3.3.2 Requirements of the Legislators

The passenger car industry has harmonized the requirements for emission-related systems in the vehicle (connector, diagnostic services, trouble codes, communication protocol, etc.) during the past two decades.

The HDVs are using two alternative communication protocols:

- ISO 15765-4
- SAE J1939/73

Both will exist in parallel for some period of time.

The United Nations Economic Commission for Europe (UNECE) World Forum for Harmonization of Vehicle Regulations (WP.29) decided to develop a global technical regulation (GTR) concerning emission-related OBD systems for HDVs and engines (UNECE GTR No 5—Technical requirements for OBD systems for road vehicles). Consequently, a single OBD protocol is required to fulfil the communication requirements of this future regulation.

The emissions control systems on highway vehicles are not the only systems with OBD capability. The non-standardized diagnostics in all other systems in the vehicle cause negative implications on maintenance and inspection procedures. This was one of the driving factors of the WWH-OBD working group to design a modular structure of the GTR such that further OBD functionalities for, e.g. safety-related systems could be added any time in the future when appropriate.

The GTR consists of a base module (general requirements) and an emission-related system module.

4.3.3.3 ISO 27145 WWH-OBD

WWH-OBD is one of the objectives of the GTR No 5. A single protocol solution is highly desired by the legislators to be developed by the automotive industry.

ISO TC22/SC3/WG1 Data Communication, SAE and JSAE (Japanese SAE) set-up a joint task force to define the principles and concept for a future single protocol solution. This activity started in early 2003 with a document from the WWH-OBD informal group called:

GENERAL PERFORMANCE CRITERIA FOR HDV EMISSION-RELATED OBD

Communication protocols

WWH-OBD meeting 6/7 November 2002—DECISION 10

This document included requirements related to the following topics:

1. Needs for a common HDV OBD Communication Protocol:
   - Today, there exist two competing communication protocols for the application of OBD to HDVs—SAE J1939 and ISO 15765.
   - In the short term, it seems that both communication standards will exist in parallel but the primary aim must be to have one common protocol, which would be to the benefit of all sectors operating under the umbrella of "the automotive industry".
2. Needs for the legislator:
   - The scope of the standard must include both current chassis control and emission control systems and must provide for the seamless addition of further control systems (both simple and complex) as the market develops.
   - The standard must offer the capability to react to the wishes of the legislator in a quick and effective manner. This particularly encompasses the following likely future requests:
     a. the standard should be extendable to passenger cars and light commercial vehicles and
     b. the standard should offer the ability to retrieve the data necessary for in-use compliance testing, e.g. to identify vehicle operation in/out of a "Not to Exceed" (NTE) zone (if NTE remains a valid concept in the future), cumulative time/distance travelled in/out of an NTE zone, operation of auxiliary control devices, torque/load readings for engine testing or to enable the use of portable emission measurement systems (PEMS).
   - The standard must include the possibility for the application of wireless communication between the OBD system and a remote interrogation unit.
   - The standard must offer the ability to retrieve in-use OBD performance data, e.g. OBD monitoring frequency, vehicle operation frequency and time of operation.
   - Clear and precise specifications within the communication protocol, with minimal variations that will result in a minimum chance of difference in interpretation that could lead to vehicles being produced that are unable to communicate fully with a generic scan tool (note: some vehicles may not actually utilize hard-wired scan-tools in the future).
   - Availability of test equipment that can verify that communication protocol specifications are being adhered to on production vehicles.
3. Needs for inspection and maintenance (I/M) testing (roadworthiness testing or roadside spot-checks):

- Clear identification of the class of each malfunction, i.e.:
  - Hierarchical safety-related malfunctions.
  - Malfunctions that result in pollutant emissions exceeding a pre-set legislative threshold.
  - Malfunctions that do not result in pollutant emissions exceeding a pre-set legislative threshold, but which could result in pollutant emissions exceeding a pre-set legislative threshold at some time soon.
  - Malfunctions that do not need to be covered by legislation but are necessary for an efficient diagnostic and maintenance function.
- Ability to communicate 'readiness' data to confirm if the vehicle is ready to be inspected and has not had its fault memory cleared recently 1 (e.g. key starts/warm-up cycles/distance travelled since memory cleared, how many diagnostics have run and been completed, ability to see if a previously active fault has been cleared by a scan tool but not fixed).
- Ability to transmit roadworthiness-related fault information (e.g. malfunction indicator (MI) status and MI commanding 'on' fault codes, odometer readings, distance travelled with MI on, emission "severity/priority" of fault).
- Ability to transmit vehicle identification information 1 (e.g. vehicle identification number (VIN), software version, odometer reading, engine ID, transmission ID, vehicle weight rating/class information).
- Ability to help combat tampering 1 (e.g. unauthorized clearing of diagnostic information).
- Ability to help identify tampered or corrupted software at the time of inspection.
- Ability to help identify (potentially) tampered hardware at the time of inspection.
- Ability to identify and retrieve roadworthiness-related information from all electronic control modules (ECM, TCM, etc.) through a single process and by wireless connection.
- Compatibility with I/M equipment ([connector], hardware, software etc.).
- Additional specific inspection needs (e.g. mode $ 08-type commands for smoke opacity test etc.).
- Compatibility with potential future telematics-based vehicle systems, e.g. bluetooth, IEEE 802.11b (or later specification).

4. Needs for the technician (i.e. repairer, replacement part maker, tool maker, etc.) are as follows:
   - Data update rates (e.g. how fast can real-time sensor data be displayed for technicians, communication speed, ability to obtain multiple PID's with single requests etc.).
   - Access to established and extendable to non-established chassis control related fault codes and real-time data in a standardized manner.
   - Mode $ 06 test results (data available in a standardized, understandable format without need to refer to a service book).

- Freeze frame data (e.g. number of frames supported, data available in the frame, usefulness of data in the frame to technicians).
- Ability to clear memory and exercise monitors (e.g. post repair) to reset readiness codes for inspection and validation of repairs.
- Cost/compatibility/upgrade potential for new and existing service tools.

4.3.3.4 The WWH-OBD Task Force also Established Requirements Related to Diagnostics and Flash Programming of ECUs.

Objectives agreed by the task force:

1. To achieve the timetable proposed and have a DIS available by January 2007.
2. To enable the separation of vehicle-level technology and communication standards from the tool-level technology.
3. Single 'off-board' protocol with a single set of services to communicate at a minimum:
   - OBD legislated data
   - Enhanced diagnostic data
   - Reprogramming

4.3.3.5 Decisions on How to Proceed with the Development Work:

1. The solution must encompass:
   - OBD legislated diagnostics
   - Enhanced vehicle diagnostics
   - Reprogramming functionality
2. There is no desire to mandate something different for OBD legislated diagnostics than that which is used for other vehicle data access.
3. The benefits this brings are as follows:
   - Consistent use of services in development will improve the quality of the protocol implementation.
   - Consistent use of services will alleviate many of the currently identified communication and data-formatting problems.
   - Reduced development cost per vehicle module.
   - De-proliferation of protocols and service sets across the automotive industry will result in fewer implementation issues.
4. The combination of these functions will pave the way for the extension of OBD diagnostics across the whole vehicle and vehicle type (HDV, medium-duty vehicle (MDV) and LDV).
5. Existing automotive and IT standards will be recognized and analysed in determining the final solution:
   - ISO 14229-1 (UDSs), ISO 15765-x and SAE J1939-x.

   - Analysis of current IT standards, e.g. for transport and network protocols to current automotive standards.
6. Recognize the need for common services and data:
   - Analyse existing data definitions from SAE J1939–21/71/73 and ISO 15031–5/6.
   - Re-use OBD legislated definitions.
   - Explore the impact of hierarchical DTC's.
7. Our vision of a 'Single Solution' encompasses the need for CAN, wireless and wired Ethernet.
8. This vision provides a modular, structured methodology to achieve and support wireless communications for OBD.
9. Our view of a 'Single Solution' across all vehicles is as follows:
   - A single communication protocol for all off-board tester applications (e.g. OBD, enhanced and reprogramming).
   - A single set of services to retrieve and download information.
   - A single set of OBD legislated data.
   - A framework for the consistent 'look and feel' for the presentation of OBD legislated data to the user.

Based on above-mentioned objectives and requirements from all parties involved an implementation concept based on existing standards was developed and published 09/2006.

ISO/PAS 27145 Road vehicles—Implementation of WWH-OBD communication requirements consisting of four parts were developed:

- Part 1: General information and use case definition.
- Part 2: Common emission-related data dictionary.
- Part 3: Common message dictionary.
- Part 4: Connection between vehicle and test equipment.

Part 1 specifies three main use cases:

- Use case 1: Information about the emission-related OBD system state—The purpose of this information package is to provide the minimum data set specified as necessary by the WWH-OBD GTR to obtain the vehicle or engine state with respect to its emission performance as specified in the WWH-OBD GTR. A typical use of this information package may be a 'roadside check' performed by an enforcement authority.
- Use case 2: Information about active emission-related malfunctions—The purpose of this information package is to provide access to the expanded data set specified as necessary by the WWH-OBD GTR to determine vehicle readiness and characterize the malfunctions detected by the OBD system. A typical use of this information package may be a periodic inspection by enforcement authorities.
- Use case 3: Information related to diagnosis for the purpose of repair—The purpose of this information package is to provide access to all OBD data required by the WWH-OBD GTR and available from the OBD system. A typical use of this

information package may be the diagnostic servicing of the vehicle or system in a workshop environment.

Part 2: Common emission-related data dictionary defines all regulatory emission-related data elements of ISO/PAS 27145. A new part may be added in the future upon availability of new legislated WWH-OBD GTR modules. The data elements are used to provide the external test equipment with the diagnostic status of the emission-related system in the vehicle. All data elements are communicated with the UDSs as defined in ISO/PAS 27145-3 common message dictionary. Data elements are DTCs, PIDs, MIDs, TIDs/routine identifiers (RIDs) and ITIDs.

Part 2 defines three (3) different sets of data elements:

- A legacy (backward compatible) data set as defined in SAE J1939-71/-73 and ISO 15031-5/SAE J1979, ISO 15031-6/SAE J2012.
- A unified data set (new data definition according to ISO/PAS 27145-2).
- A manufacturer data set (defined by manufacturer).

Part 3: Common message dictionary definition of ISO/PAS 27145 specifies the implementation of a subset of UDSs as specified in ISO 14229-1. The diagnostic services are used to communicate all diagnostic data as defined in "ISO/PAS 27145-2 Common emissions-related data dictionary".

The subset of UDSs derives from the requirements stated in the WWH-OBD GTR. The common message set defined in this part is independent of the underlying transport, network, data link and physical layer. This document does not specify any requirements for the in-vehicle network architecture.

Part 3 includes a superset of a modified version of ISO 14229-1. Several significant modifications are included in this part in order to support the data set of SAE J1939, ISO 15031-5/SAE J1979 and ISO 15031-6/SAE J2012.

Part 4: Connection between vehicle and test equipment of ISO/PAS 27145 defines the requirements to successfully establish, maintain and terminate communication with a vehicle that implements the requirements of the WWH-OBD GTR. This requires plug and play communication capabilities of the vehicle as well as any test equipment that intends to establish communication with a vehicle. This document details all the OSI layer requirements to achieve this goal.

An ISO Publicly Available Specification (PAS) requires a worldwide ballot after 3 years of publication. The outcome of the ballot was to convert and establish ISO 27145 as an international standard with the addition of a Part 5 conformance test and Part 6 external test equipment.

Figure 4.25 illustrates all referenced documents according to the OSI layers.

Two protocols are supported:

- ISO 15765 DoCAN (diagnostic communication over CAN).
- ISO 13400 DoIP (diagnostic communication over Internet protocol).

ISO 27145-1 defines the general structure of the documents and the WWH-OBD applicable use cases as specified in the PAS.

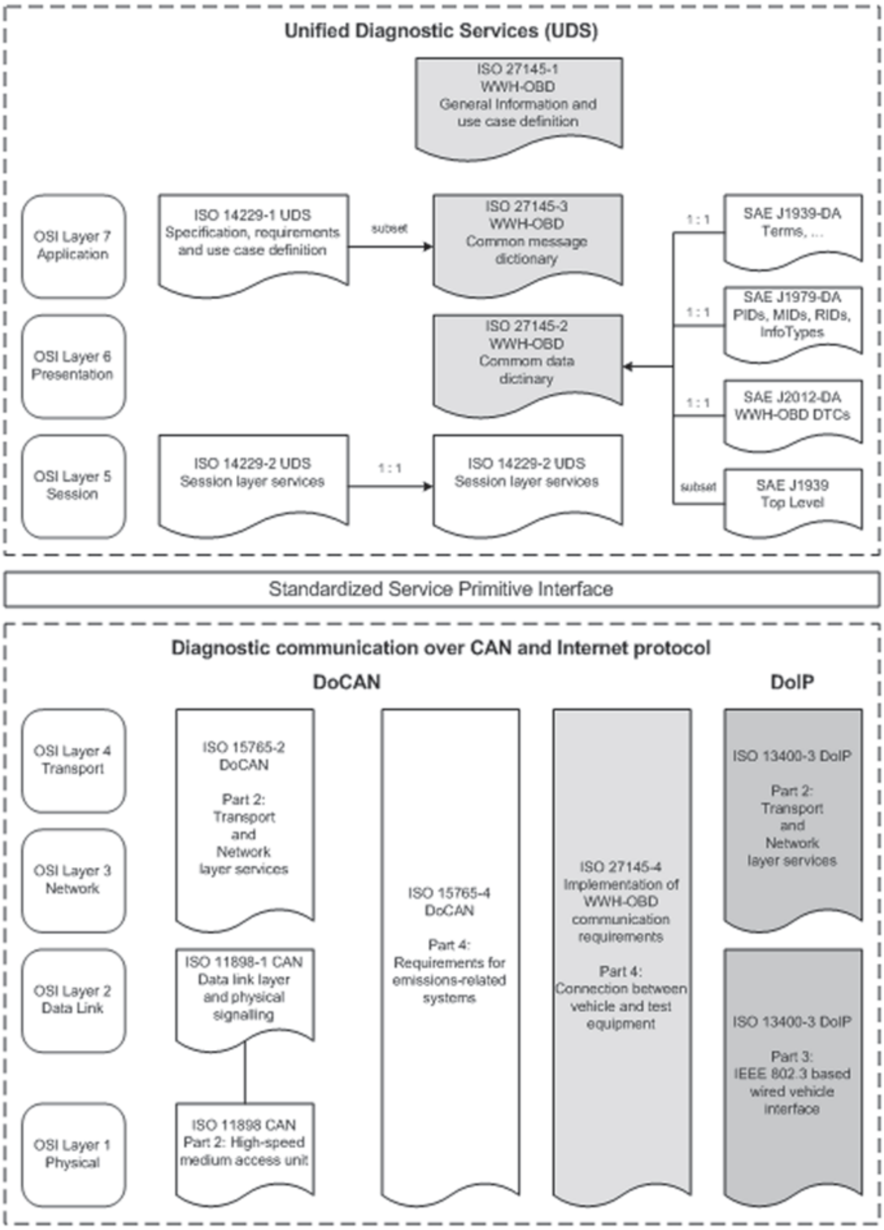

Unified Diagnostic Services (UDS)
ISO 27145-1 WWH-OBD General Information and use case definition
OSI Layer 7 Application
ISO 14229-1 UDS Specification, requirements and use case definition
subset
ISO 27145-3 WWH-OBD Common message dictionary
1 : 1
SAE J1939-DA Terms, ...
1 : 1
SAE J1979-DA PIDs, MIDs, RIDs, InfoTypes
OSI Layer 6 Presentation
ISO 27145-2 WWH-OBD Common data dictinary
1 : 1
SAE J2012-DA WWH-OBD DTCs
OSI Layer 5 Session
ISO 14229-2 UDS Session layer services
1 : 1
ISO 14229-2 UDS Session layer services
subset
SAE J1939 Top Level
Standardized Service Primitive Interface
Diagnostic communication over CAN and Internet protocol
DoCAN
DoIP
OSI Layer 4 Transport
OSI Layer 3 Network
OSI Layer 2 Data Link
OSI Layer 1 Physical
ISO 15765-2 DoCAN Part 2: Transport and Network layer services
ISO 11898-1 CAN Data link layer and physical signalling
ISO 11898 CAN Part 2: High-speed medium access unit
ISO 15765-4 DoCAN Part 4: Requirements for emissions-related systems
ISO 27145-4 Implementation of WWH-OBD communication requirements Part 4: Connection between vehicle and test equipment
ISO 13400-3 DoIP Part 2: Transport and Network layer services
ISO 13400-3 DoIP Part 3: IEEE 802.3 based wired vehicle interface

Fig. 4.25 Overview of all referenced UDS or rather WWH-OBD standards

ISO 27145-2 common data dictionary references the following documents containing emission-related data definitions:

- SAE°J1930-DA, electrical/electronic systems diagnostic terms, definitions, abbreviations and acronyms (includes all standardized terms and abbreviations).
- SAE°J1979-DA E/E diagnostic test modes (includes all standardized data items: PIDs, TIDs, MIDs, ITIDs).
- SAE°J2012-DA, diagnostic trouble codes (includes all standardized DTCs).
- SAE°J1939, top level (includes all standardized SPNs and PGNs).

ISO 27145-3 common message dictionary specifies the implementation of a subset of ISO 14229-1 UDS diagnostic services.

ISO 14229-2 UDS session layer services define the standardized service primitive interface between the OSI layer 5 (session) and OSI layer 4 (transport). Through this interface the implementation of the UDS diagnostic services is independent of the underlying communication protocols (CAN, IP). This is important when the vehicle manufacturer is required to transition from CAN to Internet protocol based on future legislation.

Table 4.18 provides an overview of the ISO 15031-5 OBD services and the mapping to ISO 27145-3 (ISO 14229-1 UDS) diagnostic services and associated subfunctions and data ranges.

4.4 SAE J1939

SAE J1939 is a set of standards for both in-vehicle normal ECU to ECU communication protocol and diagnostic communication protocol. The standards cover relevant OSI layers and specify physical link (cable), how the messages are built up, NMT, in-vehicle communication with data items, diagnostic communication with DMs and data, name claiming and conformance test specification. Parts of the set of standards are used in heavy-duty and medium-duty applications worldwide.

When SAE J1939 was introduced, CAN was not mentioned but it was soon included. First, the -71 layer was introduced for normal in-vehicle communication and later the J1939-73 diagnostic layer was introduced.

The protocol is used in HDVs including trailers, agricultural machines, off-road equipment, boats and stationary engines and it has been discussed for residential vehicles also. The main reason is that the protocol has been implemented in engine control modules (ECMs) for medium-duty/heavy-duty diesel engines as a standard.

The situation in the USA is that a truck may be a chassis from an OEM, with an engine from an engine manufacturer and a transmission from a transmission supplier. It is possible for the customer of the vehicle to equip the vehicle with systems from different suppliers. A fleet manager maybe have different truck brands (e.g. Navistar, Volvo and Freightliner) but wants to keep the same engine manufacturer (e.g. Cummings) for the complete fleet and the same for the different systems, e.g. transmission and brakes.

Table 4.18 Mapping emission-related system OBD with WWH-OBD GTR

OBD SID	ISO 15031-5/SAE J1979 service	WWH-OBD SID, SFID	ISO 27145-3/ISO 14229-1 UDS service name	ISO 27145-3/ISO 14229-1 sub-function name/data range (all data are referenced: SAE J1979-DA, J2012-DA)
0×01	Request Current powertrain diagnostic data	0×22, 2-byte DID (3 PIDs max.)	ReadDataByIdentifier	PIDs: 0xF400–0xF5FF (low byte=PID#)
0×02	Request powertrain freeze frame data	0×19, 0×04, 3 byte DTC#, Frame# 0×19, 0×06, 3 byte DTC#, ExtRecord#	ReadDTCInformation	reportDTCSnapshotRecordByDTCNumber reportDTCExtendedDataRecordByDTCNumber
0×03	Request emission-related diagnostic trouble codes	0×19, 0×42, FGID, DTCStatusMask, DTCSeverityMask	ReadDTCInformation	reportWWHOBDDTCByMaskRecord FGID=FunctionalGroup ID (e.g. emissions, safety, …)
0×04	Clear/reset emission-related diagnostic information	0×14, groupOfDTC	ClearDTCInformation	groupOfDTC=3 byte DTC 0xFFFFFF for all DTCs
0×05	Request oxygen sensor monitoring test results	0×22, 2-byte DID (MID)	ReadDataByIdentifier	MIDs: 0xF600–0xF7FF (low byte=TID# of monitor)
0×06	Request on-board monitoring test results for specific monitored systems	0×22, 2-byte DID (MID)	ReadDataByIdentifier	MIDs: 0xF600–0xF7FF (low byte=OBDMID#)
0×07	Request emission-related DTCs detected during current or last completed driving cycle	0×19, 0×42, FGID, DTCStatusMask, DTCSeverityMask	ReadDTCInformation	reportWWHOBDDTCByMaskRecord FGID=FunctionalGroup ID (e.g. emissions, safety, …)
0×08	Request control of on-board system, test or component	0×31, 0×01, 2-byte RID	RoutineControl	startRoutine, Routine ID (e.g. 0×01, 0×02, …) 2-byte RID (low byte=TID of service 0×08)
0×09	Request vehicle information	0×22, 2-byte DID (ITID)	ReadDataByIdentifier	ITIDs: 0xF800–0xF8FF (low byte=InfoType#)
0×0A	Request emission-related DTCs with permanent status	0×19, 0×55, FGID	ReadDTCInformation	reportWWHOBDDTCWithPermanentStatus FGID=FunctionalGroup ID (e.g. emissions, safety, …)

SAE J1939-71 makes the in-vehicle communication between the different ECUs possible, no or small (e.g. tuning of ECU addresses) adaptions are needed.

The ECUs are more or less "of the shelf".

The same situation for passenger car is that every OEM sets up his own network protocol and all ECUs must be adapted to co-exist in the vehicle.

4.4.1 Structure of SAE J1939

The structure of SAE J1939 is the same as the OSI layers. The following standards will be focused on in this document: SAE J1939-21, SAE J1939-71 and SAE J1939-73.

The lowest layers (SAE J1939-11 and SAE J1939-15) are almost similar to ISO 11898, i.e. -15 specifies an unshielded twisted cable and -11 specifies a shielded twisted cable and the CAN bus speed is currently 250 kbps and the CAN identifier length is 29 bits.

SAE J1939 is working on extending the CAN bus speed to 500 kbps.

SAE J1939-13 defines the connector, which is a nine-pin round Deutz connector with two pins for CAN and two pins for SAE J1708 which is the physical layer of SAE J1587. It is not allowed in the USA to use the SAE J1962 ("ISO" or D-shaped) connector together with SAE J1939 protocol. This specific requirement does not exist in Europe and Volvo truck and Volvo bus use SAE J1939 together with the D-shaped connector.

The usage of J1939 is for controlling vehicle or engine application. SAE J1939-71 is commonly used in heavy-duty applications for at least powertrain applications. Some OEMs use it also for complete vehicle applications.

There is an in facto agreement in the USA to use SAE J1939-73 for legislated OBD diagnostics, and some OEMs use the same protocol as an enhanced protocol for the complete vehicle and have implemented services for software download and everything which is needed for workshop fault tracing and repair.

Trucks in the rest of the world usually have adopted an ISO protocol, usually ISO 14230 ("Keyword Protocol (KWP) 2000 on CAN") or ISO 15765-3 (also known as ISO 14229-3 or DoCAN).

The ISO protocol does not interfere with J1939-71 which is used for vehicle control.

It is not allowed to implement both SAE J1939-73 and ISO 15765-4 for legislated OBD protocol. The reason is that independent scan tools will utilize an algorithm to detect as to which type of legislated OBD protocol is implemented in the vehicle. The algorithm is based on scanning through all allowed protocols and the tool stops when it has detected the first protocol.

It would be too advanced for the tool to try to use two completely different protocols and combine the data.

Table 4.19 OSI layers as a function of the SAE J1939

Application layer	SAE J1939-71	SAE J1939-73
Presentation layer	–	–
Session layer	–	–
Transport layer	SAE J1939-21	–
Network layer	SAE J1939-31	–
Data link layer	SAE J1939-21	–
Physical layer	SAE J1939-15	SAE J1939-11

The SAE J1939-73 is allowed for legislated OBD communication for US 10 emission legislation, Euro IV, Euro V and the upcoming Euro VI emission legislations.

Volvo is probably the only OEM in Europe which has implemented SAE J1939-73 as legislated OBD protocol for Euro IV and Euro V emission legislations. The company will transfer to ISO protocols in the future (Table 4.19).

4.4.2 SAE J1939-21 Data Link Layer

The SAE J1939-21 defines how the PDU is built up.

A SAE J1939 PDU consist of 3 bit priority (P), 1 reserved bit (R), 1 bit for data page (DP), 8 bit for PDU format (PF), 8 bit for PDU specific (PS) and 8 bit for source address (SA) plus up to 64 bit of data (8 byte) (Fig. 4.26).

- The priority bits set the priority during arbitration and 0 is the highest priority, 7 the lowest. A recommended priority is assigned to all PGNs listed in the standard, but the receiver should ignore the priority bits; this is due to the fact that the priority may be changed.
- Reserved bit is not the CAN reserved bit, but reserved for future expansion of the standard.
- Data page: All PGNs must be assigned to page 0 before page 1 is used.
- PDU format, PF, is used to determine the PGN.
- PDU specific, PS, can be either the destination address or a group extension. If the PS is below 240 then it is a destination address, otherwise it is a group extension.
  - Destination address: It specifies the ECU (or address) that should listen to the message. 255 is a global address for all ECUs.
  - Group extension: It provides 4,096 data groups per DP plus the 240 extra PDUs (PS < 240). In total, there are (4096 + 240)*2 possible data groups.
- Source address is the ECU sending the message. The addresses are defined in SAE J1939-81.
- PGN is based on reserved bit, DP bit and then 16 more bits.
- Data field: up to 8 byte of information in a single frame. Non-used data bits should be set to non-available (padded to '1'), which would mean in practice that

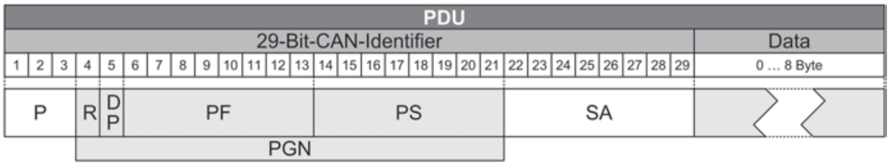


Fig. 4.26 SAE J1939-21 PDU and the CN-identifier assignment

the CAN protocol will include bit stuffing and extend the number of bits in the message.

Some messages are longer than one single CAN frame (8 byte), e.g. VIN which consist of 17 ASCII characters. When a long message (up to 1785 bytes) should be transmitted, it is possible to send the message as a segmented message using the transport protocol function and there are two methods:

4.4.2.1 Transport Protocol (TP)

Method 1: Broadcast Announce Message, TP_BAM TP_BAM: A message with a global address, which means that all ECUs listen to the message. The message starts with a Connection Management (CM) message, PGN 00EC00 with a control byte indicating TP_BAM and then the PGNs with an inter-frame time of minimum 50 ms. This method should not be used if it is not specified in the applicable standard (i.e. SAE J1939-71, SAE J1939-73 or SAE J1939-03). The main reason is that all ECUs have to listen to something which may be a message between one ECU and a scan tool and therefore spend resources on a message which does not concern them.

One of the examples when a TP_BAM could be used is during scan tool initialization before the tool knows which ECUs are installed in the vehicle. It can at that time broadcast a message in order to identify all that support the service or data.

To identify if a vehicle is utilizing SAE J1939-73 as a legislated OBD protocol, the DM 5 (Readiness 1) is used (Fig. 4.27).

Method 2: Connection Management, TP_CM The other method is called TP_CM: A message is sent from point to point. The sending ECU sends a CM message indicating Request to Send (RTS). The receiver responds with a Clear To Send (CTS) with the number of packets (buffer size) it may accept and the sequence number of the expected packet.

The parameter group, together with the data is then transmitted in several data transfer messages (DT), wherein the first byte indicates the sequence number in each case. It is possible to pause the communication and to abort. This method is the preferred one when it comes to diagnostic communication to an off-board client (scan tool) (Fig. 4.28).

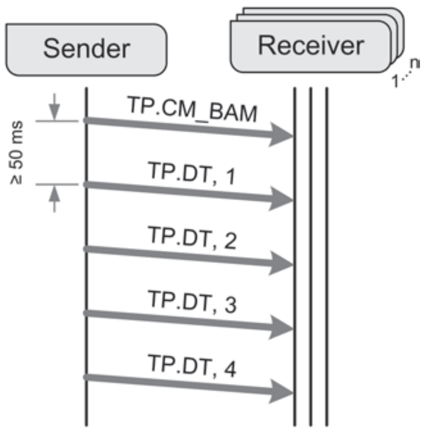


Fig. 4.27 J1939 transport protocol—broadcast announce message

Request: Normal in-vehicle control data are sent periodically on the bus but sometimes some data are needed and they are not usually sent on the bus. It is possible to request the data (or PGN) in those situations. One example is VIN which is used to identify the vehicle by a scan tool. The VIN is 17 characters and there is no need to send it periodically, so the scan tool needs to request the data from the ECU which has the information.

Data: The ECU either sends a negative acknowledge, NACK, if it does not have the data or respond to data. If one data element is not used in the PGN, then the bits for that suspect parameter number shall be set to 1.

The response time is 200 ms but the requesting equipment needs to wait up to 1.25 s before it times out. The main reason is that bridges (gateways or routers) can delay the message.

4.4.3 SAE J1939-31—Network Layer

The SAE J1939-31 network layer standard defines how a complete network should be designed. The standard describes gateway and router functionalities (Fig. 4.29).

Gateways can be within the vehicle to isolate different buses from each other (e.g. one private network for brake system, another network for the cab controller, a third for powertrain) or to act as bridges between, e.g. a tractor and a trailer.

4.4.4 SAE J1939-71 71—Vehicle Application Layer

SAE J1939 has relationship to SAE J1587 which is usually implemented on SAE J1708 bus.

J1939-71 defines signals for normal communication, point-to-point in-vehicle communication. The signals are defined as SPNs.

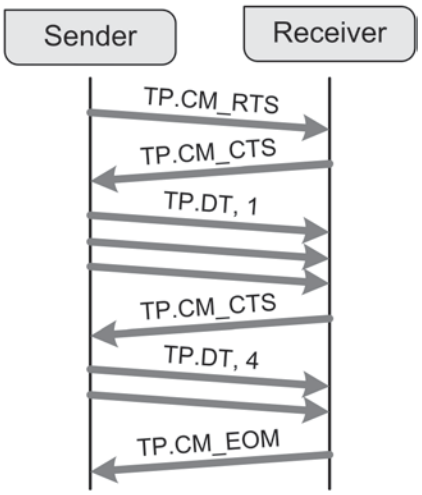


Fig. 4.28 J1939 transport protocol—connection management

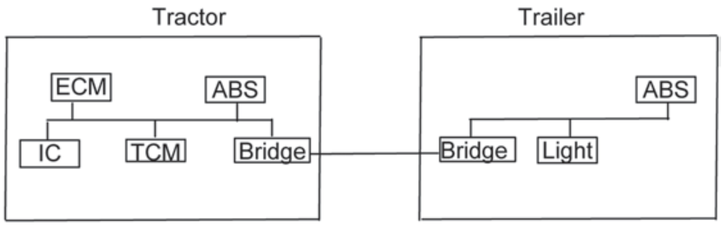


Fig. 4.29 Connection between tractor and trailer via network bridges

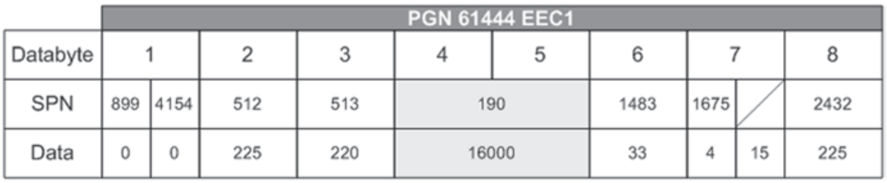
The SPN is a fictive number which usually only exists in the standard and is not seen on the bus, except for some services, e.g. DM 24 which reports which SPNs are implemented as data stream, freeze frame parameters[5] or test values.[6]

The SPN is defined with length and scaling information and is connected to a PGN, which is more or less a part of the CAN identifier for the message.

Some SPNs, e.g. SPN 237 = VIN, are longer than a CAN frame and must be sent as a segmented message.

[5] Freeze frame is data which has have been frozen at the occurrence of the detection of a fault. The intention is that the information in the freeze frame may help the service technician to reproduce the conditions which existed when the malfunction was detected. Example of a freeze frame parameters are engine speed and ambient air temperature. The electronic control module will store these two parameters when it detects a malfunction.

[6] Test value is the value which is compared to the fault limits of a monitor to judge if the latest evaluation was pass or fail. It can be seen as an analogue value of a fault (diagnostic trouble code), e.g., if the test value is 0 × 8340 and the fault limit is 0 × 8000 then there should be a fault code stored in the electronic control module.



	PGN 61444 EEC1									
Databyte	1		2	3	4	5	6	7		8
SPN	899	4154	512	513	190		1483	1675		2432
Data	0	0	225	220	16000		33	4	15	225

Fig. 4.30 Example for number-dedicated SPNs

The SPN can be sent either on request (as for diagnostic information) or as a periodic transmission.

The request of an SPN is done by requesting the PGN and then decoding the SPN from the data field of the CAN frame (Fig. 4.30).

This specific PGN (EEC1) is transmitted as a function of engine speed, i.e. low engine speed = low transmission frequency, high engine speed = high transmission frequency. The SPN for engine speed is 190 and the value of SPN190 is 16,000.

In order to translate the information, which is received as the data field in a CAN frame, it is just to copy the content of data byte 4 and 5 and multiply it to 0.125 and the engine speed is given in rpm. The definition of EEC1 and the different SPNs can be found in SAE J1939-71.

The other SPNs in EEC1 are 899 (Engine Torque Mode), 4154 (Actual Engine Per cent Torque High Resolution), 512 (Driver's Demand Engine Per cent Torque), 513 (Actual Engine Per cent Torque), 1483 (Source Address of Controlling Device for Engine Control), 1675 (Engine Starter Mode) and 2432 (Engine Demand Per cent Torque). The value of 15 in data byte 7 indicates "no information".

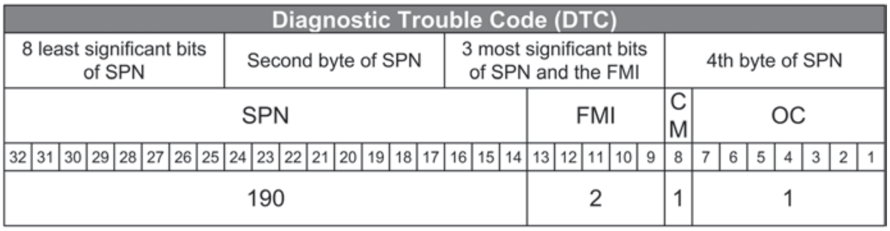
4.4.4.1 SPN and Fault Information

Since the main purpose of SPN is for in-vehicle communication, i.e. from ECU A to ECU B, then a fault information concept has been developed. The valid data range for a 1-byte SPN is 0–250 and 0xFE is used to indicate that the source of the signal cannot be trusted due to a fault and 0xFF indicates that the signal is not updated. It is up to the receiving ECU to decide which default action it should take. It could be that the latest valid data are used or a default value, e.g. 20 degrees as ambient air temperature, is used in the application.

The method of using 0xFB–0xFF can be useful for normal in-vehicle communication, but it makes it impossible to store raw or un-defaulted data in a freeze frame or to report un-defaulted data stream data to a scan tool.

Un-defaulted data are defined as the data after linearization and converted to an engineering unit, e.g. the ADC measures voltage and uses a look-up table to convert the voltage to a temperature.

The defaulted data are when a malfunction is detected and the value of the parameter is changed to, e.g. 20 degrees, or, if sent on CAN, to 0xFE to indicate to



Diagnostic Trouble Code (DTC)																															
8 least significant bits of SPN								Second byte of SPN								3 most significant bits of SPN and the FMI								4th byte of SPN							
SPN																			FMI					CM	OC						
32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
190																			2					1	1						

Fig. 4.31 Example for SPN 190 trouble code (engine speed)

another ECM that the signal is not valid. If the defaulted value is stored in a freeze frame, then the data will not help the service technician, i.e. he/she will not know at which temperature the malfunction was detected.

4.4.5 SAE J1939-73—Diagnostics

Diagnostic communications are based on SAE J1587 and include special PGNs called DMs. They can be considered as equal to diagnostic services in ISO 15031-5 or ISO 14229-1. Most of the DMs are sent on request. The main exception is DM1 which is used as communication message to the instrument cluster in order to illuminate warning lamps. If the ECU does not support the request DM, then it will respond with a NACK. The DTC contains the 19-bit SPN with a 5-bit failure mode indicator (FMI) (Fig. 4.31).

SPN is the suspect parameter number of the failed component. FMI is the failure mode indicator (comparable to failure type byte of ISO 14229-1). The CM is a conversion method, since there has been at least four different methods on how to represent the DTCs. At the moment there is just one method, #4 Intel format with 19 bit SPN + 5 bit FMI and CM0, which is allowed for legislated OBD communication. The other versions shall not be used. Finally, OC is an occurrence counter, which counts the number of times the fault has been gone from an InActive (previously Active) state to an Active state.

This method of using the signal as the part of the DTC is useful for simple diagnostic function, e.g. if an ambient air temperature sensor detects an electrical fault, then it will be clear for the service technician on which signal he should check or read. As said, this will work fine as long as there is a one-to-one relationship, i.e. the monitor is based on one signal/sensor. OBD II usually requires more advanced functions, e.g. the catalyst monitor for a gasoline engine maybe utilize the intake manifold airflow sensor and the rear oxygen sensor plus some other sensor values to check if the conversion ratio in the catalyst is OK. This is a multiple sensors to one DTC scenario and it is not possible to read a single sensor anymore and the SPN is more "fictive". The detection mechanism of a fault is usually called a monitor.

4.4.5.1 Fault Codes

The first DM for reading fault codes is DM1 to read active DTCs.

A DTC is active when it is detected after debounce filtering and the service is intended mostly to inform the instrument cluster to illuminate an appropriate lamp, red, yellow or white lamp. The DM1 service is sent periodically to the instrument cluster. It includes 4 bits, 1 bit per warning indicator, i.e. malfunction indicator lamp, red stop light, yellow warning and white information light. After the lamp information, the fault codes are sent.

DM2: Previously active DTCs. The service is sent on request and includes the warning light information plus all DTCs which are no longer detected as active faults.

The states for emission-related DTCs are more complicated after US10:

First, the DTC will show up as a pending DTC (DM6) in the first failed driving cycle.

If the monitor fails in the next consecutive driving cycle, then it will transit to an emission-related active DTC (DM12) and will not be reported in DM6. When the DTC heals, it will transit to previously active emission-related DTCs (DM23) and it will disappear completely after 40 fault-free warm-up cycles.

The DTC will also be reported as a permanent DTC (DM28) as long as the MI is commanded on.

N.B. It is allowed to report a DTC as both pending and emission-related active DTCs according to the standard, but any emission-related ECU which does will not fulfil California legislation (Table 4.20).

A note must be mentioned regarding DM24 and DM25.

DM24 reports which SuspectParameterNumbers are implemented as data stream parameters, as freeze frame parameters and as test values. It is not clear which SuspectParameterNumbers should be reported as data stream parameters: All according to legislation, all according to what is sent on the bus or those SuspectParameterNumbers that are broadcasted on request? The service is clearer on freeze frame parameters since the freeze frame is well defined.

Requesting latest test value from a non-continuous executing monitor is the same as Service $ 06 of ISO 15031-5, but first the DM25 is read and the SuspectParameterNumbers for test results are reported in the response.

The SuspectParameterNumber and the correct FailureModeIndicator (which is not reported by DM25) must be put into the DM7 request. It may also be possible to request FailureModeIndicator = 31 which means all FailureModes.

The response is sent by the ElectronicControlUnit as a DM30 response including the SuspectParameterNumber and the FailureModeIndicator, followed by a SLOT identifier which includes length and scaling information for the next parameter, the TestValue. The SLOT identifier should always report that the TestValue has a length of 2 bytes in DM30.

The TestValue is the value which the specific monitor compares with the fault limits to judge if the result from the monitor is a pass or fail. The TestLimits are also reported in DM30. The intention of DM7/DM30 is to show how close to the fault limit the monitor is, e.g. if the TestValue is 90 % of the test limit, then the monitor

Table 4.20 Relation of emission-based diagnostics standards

	ISO 15031-5	ISO 27145-3	SAE J1939-73 US10	SAE J1939-73 EURO VI
Request data stream	Service $ 01	Service $ 22	SPNs + DM21, DM5, DM 26, DM32, DM33, DM34	
Request freeze frame	Service $ 02	Service $ 19 04	DM25	
Request pending DTCs	Service $ 07	Service $ 19 42	DM6	DM41, DM44, DM47, DM50
Request confirmed DTCs	Service $ 03	Service $ 19 42	DM 12	DM42, DM45, DM48, DM51
Request permanent DTCs	Service $ 0A	Service $ 19 55	DM28	
Request previous DTCs		Service $ 19 42	DM23	DM43, DM46,
DM 49, DM52				
Clear DTCs	Service $ 04	Service $ 14	DM 11	DM11
Request test results	Service $ 06	Service $ 22	DM7, DM30	DM7, DM30
Command test	Service $ 08	Service $ 31	DM8	DM8
Request infoType	Service $ 09	Service $ 22	DM 19, DM20, +SPN	DM 19, DM20, +SPN

Note: ISO 15031-5 (SAE J1979) is allowed for legislated OBD communication for HDVs in the USA and for HDVs until Euro VI legislation in Europe. It uses Service $ 01–0A for retrieving the legislated communication. ISO 14229-1 (ISO 15765-3 DoCAN) is used for enhanced diagnostics in both passenger cars and some HDVs. The Euro VI column shows that there is a different set of DMs for fulfilling Euro VI legislation

could detect a failure during some conditions. The TestValues will usually differ each time the monitor is executed since they are affected by component deviation, component ageing and also driving and environmental conditions.

4.4.5.2 Scan Tool Initialization

The scan tool initialization procedure is easier than for a scan tool for ISO 15765-4 since there is currently only one allowed bus speed, 250 kbps and all Electronic-ControlUnits shall utilize 29-bit CAN identifiers.

First, the tool will check for PGN 61444 EEC1 which is always transmitted if there is an engine ElectronicControlUnit on the network. Then it will send a DM5 Diagnostic Readiness 1 request as a BAM request. All ECUs that utilize SAE J1939-73 will respond with the information of, e.g. the number of active DTCs, the number of previously active DTCs, the monitor groups that have executed since last clear DTC and the OBD compliance (similar to PID $ 1C of ISO 15031-5).

It is possible that vehicles prior 2010 in the USA do not utilize SAE J1939-73 for diagnostic communication, and in Europe it is quite common to use SAE J1939-71 together with ISO 15765-4/ISO 15031-5 for OBD communication. It is therefore not enough to check the ParameterGroupNumber 61444 EEC1 to distinguish if the vehicle utilizes SAE J1939-73.

The scan tool will build a list of the ElectronicControlUnits that has responded to this request and the OBD compliance will be an input of the services that are implemented. Table 1 of SAE J1939-73 states which DMs and which information are needed to comply with different OBD legislations. It is not clear how to handle smart sensors or smart actuators since these simple devices have a limited implementation of the diagnostic protocol. They have maybe only implemented DM1 active DTCs, DM11 clear DTCs, DM 19 CALibrationIDentification/Calibration-VerificationNumber and DM5 for scan tool initialization, but the vehicle needs to comply to FinalRegulationOrder 1971 which is for US10 legislation in California.

4.4.6 SAE J1939-81 81—NMT

SAE J1939-81 includes NMT, i.e. how to handle if two ECUs are connected to the same network with the same ECU address. Normally two ECUs shall never share the same SA, but in some type of vehicles, this may happen, e.g. a tractor with two trailers with both trailers having a brake system.

It uses a NAME field and a method to claim addresses. If there are multiple ECUs with same address, then the ECU with the lowest NAME will win and the others have to claim other addresses. The claiming can be seen as an ECU address arbitration. This works as the best in theory but there are problems in real life; the ECUs detect bus-off before they could claim the address, they misunderstand the response since maybe both ECUs respond to the same Name claim response and get the same response, i.e. they cannot know if the data are intended for the own ECU or the other ECU and both will react as for communication faults.

4.4.7 SAE J1939-84

The standard is a conformance test, but not equal to SAE J1699-3 for LDVs. The J1699-3 is a conformance test for a complete vehicle and J1939-84 is a very thin test specification for a single ECU. The implementation of the standard can also be downloaded as a Dynamic Link Library (DLL) on www.sourceforge.org. The standard is used to verify that the application in the ECU fulfil the SAE J1939-73 standard.

4.5 CanKingdom

CanKingdom (CK), first published in 1990, is considered the ancestor of the CAN-based higher layer protocols (HLP). In many respects, it is quite different from later CAN HLPs:

- It is not really a HLP; it is a meta-HLP, i.e. a protocol for constructing a protocol.
- It is not a communication protocol, but a system control protocol.
- It is designed to maximize the composability of a system.
- It is designed for achieving high performance at low cost throughout the life cycle of a system.
- It is intended for hard real-time and safety critical systems.
- It is designed to allow a mix of time-triggered, event-triggered and sequential schedules.
- It is designed to minimize development time of systems and modules by separating the system design and module design tasks as far as possible.
- It is designed to allow a system to be individually optimized at any time during its lifetime.
- It is designed to allow changes between modes during run time, e.g. from normal mode to a limp home mode.
- It is not compliant with the OSI model.
- CK uses a unique vocabulary for essential terms to make them unambiguous.

CK is based on an unorthodox approach to Distributed Embedded Control Systems. Most CAN systems are based on the seven-layer OSI model, where the different applications in a network are separated by an independent communication layer. However, this approach can lead to severe timing and synchronization problems because the timing of CAN messages is not controlled. A great advantage of CK is that it allows the system designer to take full control of message transmissions by scheduling them in time or sequence, as well as having unschedulable messages (as alarms) to be transmitted when needed. In the approach for CK the whole system is viewed as a combination of devices, such as joysticks, actuators, sensors, etc., all controlled by one imaginary application. This imaginary application is broken down into a number of real sub-applications, with each sub-application residing in a module of its own and integrated into the respective device. Sub-applications have two parts: A local part takes care of everything needed for the device it resides in and the other part interacts with other devices. In this way, we have two clear layers: an imaginary system layer and a module layer. The imaginary system layer is brought to reality by a third "glue" layer that integrates the sub-applications with each other. This glue layer provides a common application programming interface (API) that is partly based on a serial communication. Thus, the communication is not a separate layer as in the OSI model; it is an inherent part of the system, gluing sub-applications together to run as one common real-time application. The restrictions imposed by the serial bus communication make the interfaces between the respective sub-applications very clean and predictable in time and sequence.

CAN is the serial communication of choice and the basis of the glue layer in CAN Kingdom. Each module has primitives of the glue layer that are completed by information sent during a configuration phase. A specific system module containing all necessary information needed to harmonize each module to the system requirements controls this phase. In this way, generic modules can be easily combined into complex, high-performance systems. In a simple system, the system module can be disconnected after the setup procedure. However, in more advanced systems, the system module is an integrated part of the system, taking a monitoring and supervising role during run time, allowing for hot swaps of modules, changing of modes, etc. CK can be implemented in a modular way. A full implementation requires roughly 5.5 K flash and 100 K RAM.

4.5.1 Background

In the early 1980s, the Swedish company Rovac developed an advanced factory automation system based on distributed embedded controllers. The whole factory was seen as one big application that included robots, material supply, mould positioning, heat control, etc. This application was broken down into as small pieces as possible and each piece was executed in a separate micro-controller, physically integrated into the device it controlled or monitored. As an example, a robot was constructed of a set of actuators connected by structural parts, tubes and swivels. In this way, special robots could be easily designed using standard parts. The micro-controllers were grouped according to their functions and the members of each group were connected to each other and to a central computer by a serial communication bus. The central computer coordinated and supervised the groups to act in concert in a safe manner. The concept, designated 'Trainet', turned out to be very flexible and efficient, but the bit rate of the communication, 9,600 bps, limited the update frequency to 25 Hz, i.e. any feedback control loop had to be executed locally.

When CAN became available in 1988, high-speed bus communication was available at a reasonable cost. As the concept of CAN is to minimize the need for run-time bandwidth by defining as much as possible off-line, it was a perfect match with the Rovac ideas. The combination of Trainet features and CAN resulted in CK, first published in 1990 by the company Kvaser. It was developed further and version 3 was made available at CiA 1992 and reached a broader audience. CK v. 3 formed the basis for the US DoD CDA 101 project for a common CAN-based protocol for different types of airborne and seaborne targets. During the development period of CDA 101, from 1996 until 2001, CK was further improved to meet the high requirements for any aspect of an embedded CAN, such as adding improved support for hard real-time and synchronization, composability, membership verification, safety, troubleshooting, etc.

Being a meta-HLP, CK is used as a base for proprietary HLPs, e.g. the Mercury "SmartCraft" for pleasure boats and the Sauer–Danfoss "Plus+1" for off-highway machines.

4.5.2 The Concept Behind CK

A cornerstone of the concept is the notion of a system: An electro-mechanical system is constructed of a number of modules, each with an ECU that is connected to a serial communication of some kind. Each module has a specific role in the system, e.g. a steering wheel or joystick, a gearbox, a motor, an actuator, a sensor, etc. They are all connected as nodes in a system network. A system designer has to combine a number of modules and make them perform in concert.

In this way, we have three concise layers:

- System layer
- Glue layer
- Module layer

The system is also seen as one big application, broken down into sub-applications that reside in respective nodes and are coordinated by the glue layer. The glue layer can then be seen as an API built upon a specific serial communication protocol. The qualities of the glue layer are therefore highly dependent on the qualities of the chosen communication protocol.

This concept has many advantages:

- The glue layer makes a clean and simple interface between interacting modules as it is only control and data messages that are transmitted and received according to need.
- Module designers can concentrate on the performance of the module and need not know much about the system.
- The system designer has only to see to it that each module receives and transmits messages as needed to perform in concert with the other nodes and does not need to know much about each ECU.
- The performance of each module can be easily checked individually as a stream of messages can be used to simulate the rest of the system. If the module responds to commands and is able to transmit messages according the specification, it is OK.
- Modules and systems can be developed in parallel, saving time and money.

This concept does not fit the OSI model. The OSI model is based on a concept where modules just need to exchange information and do not require any form of coordination between communication sessions.

CAN is very well suited as the basis for a glue layer:

- It conveys 11–93 bits at a time from one node to all other connected nodes in a safe, predictable way.
- Any bit rate between 10 kbps and 1 Mbps is supported.
- It can be used for any scheduled and/or unscheduled transmissions.
- Time scheduled, sequence scheduled and unscheduled messages can be transmitted simultaneously with a guaranteed latency time.

4.5.3 Overview

CK is a glue layer based on CAN. It contains a set of rules—all in all 18—that separate the module layer and system layer as much as possible. Only three of these rules are mandatory (Table 4.21).

Table 4.21 Rules of the CanKingdom glue layer

Rule	Description	
1	Start/stop modes. To force a module to stop and go into silent mode	Mandatory
2	Initiate. To establish an exclusive communication between a module and the configuration tool	Mandatory
3	Assign CAN IDs to receive and transmit data in a module	Mandatory
4	Assign groups. Make a module a member of a group to receive group commands	–
5	Remove groups. To expel a module from a group	–
6	Trigger setting. To make a module trigger a task on a message or an event	–
7	Assigning modules to product or producer-specific groups	–
8	Assigning a physical address to a module identified by its serial number	–
9	Change the physical address of a module	–
10	Bit timing register setting	–
11	Inhibit time. To prevent a module from retransmitting a message until a certain time has elapsed	–
12	Circular time base setup. To create a global clock	–
13	Repetition rate and open window setup. To set up a time-triggered communication	–
14	Giving common system wide identifications to messages or groups of messages	–
15	Create CAN messages from local parameters	–
16	Create CAN messages where the data field is extended into the ID field	–
17	Creating bit filter masks	–
18	Creating advanced message filters	–

A full implementation of CK requires roughly 5.5 K flash memory and 100 K RAM.

4.5.4 CK Vocabulary

In order to make the CK rules and functions unambiguous, it uses a unique vocabulary and specific CK terms are spelt with a capital letter. The description is based on a simile of a kingdom where the King in his Capital sets the rules for the Kingdom. The Kingdom has Cities, each of them ruled by a Mayor. The Kingdom is designed by a Kingdom Founder, i.e. the system designer and Cities by City Founders, i.e. module designers. Any information exchange between Cities in the Kingdom is made via a Postal System. The Capital and each City has a Post Office with a Postmaster (a CAN controller) (Fig. 4.32).

The only way to communicate within a Kingdom is to use Letters (CAN messages). Each Letter has an Envelope (CAN ID) and a Page (CAN data field). A Page is built up of 0–8 Lines (bytes in the CAN data field) and a Line can be constructed of 0–8 Dots (bits in a byte in the CAN data field). Pages are organized in Docu-

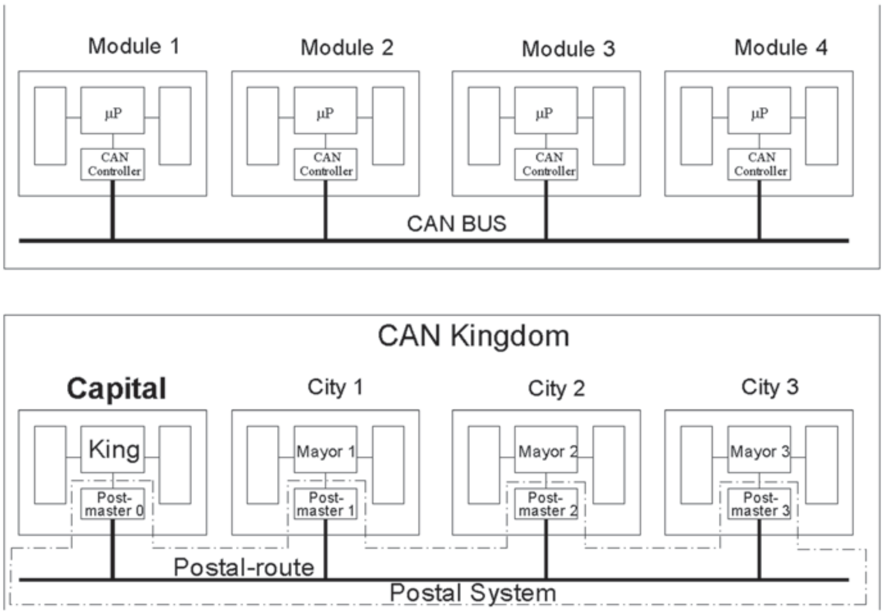


Fig. 4.32 Structure of CanKingdom

ments. The Document is the key for the Cities to encode or decode the Pages. A Document can contain one un-enumerated Page or more enumerated Pages. Cities have matching Documents for coordinated tasks; one is set up for transmission and the other ones for reception. The King uses a King's Document to configure each City. In this process, he assigns Envelopes to matching Documents. Then no CAN ID (except the one for the King's Document) is predefined and a system designer is free to give any message its proper priority. A Document can contain not only data but also tasks, e.g. a Letter with a blank Page or even a Letter for another City can be used for triggering the execution of tasks in one or more Cities. A programmer may see the transmission entities as threads (Fig. 4.33):

4.5.5 King's Document

The King's Document contains many Pages, one for each rule. The Kingdom Founder has to implement all King's Pages supported by any City he/she will use in his Kingdom in order to set up each City in a proper way. The King's Document contains then at least three Pages (the mandatory rules) but also any Pages corresponding to additional rules implemented by selected Cities. All King's Pages use the Envelope 0 (CAN ID 0 Std) as default. (This number can be changed if neces-

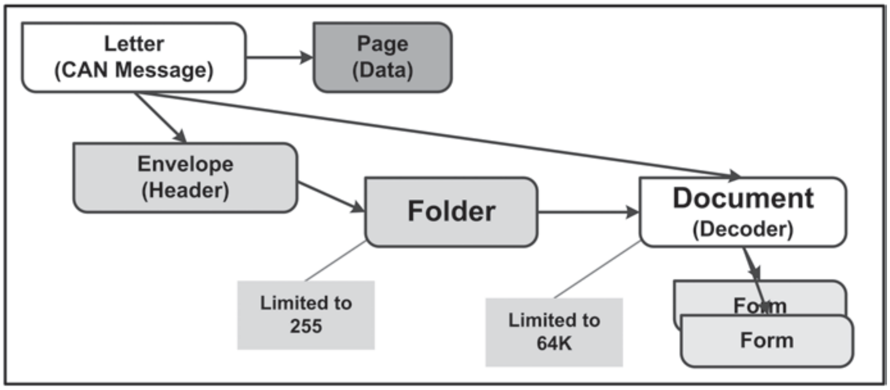


Fig. 4.33 Organizational structure of the CanKingdom messages

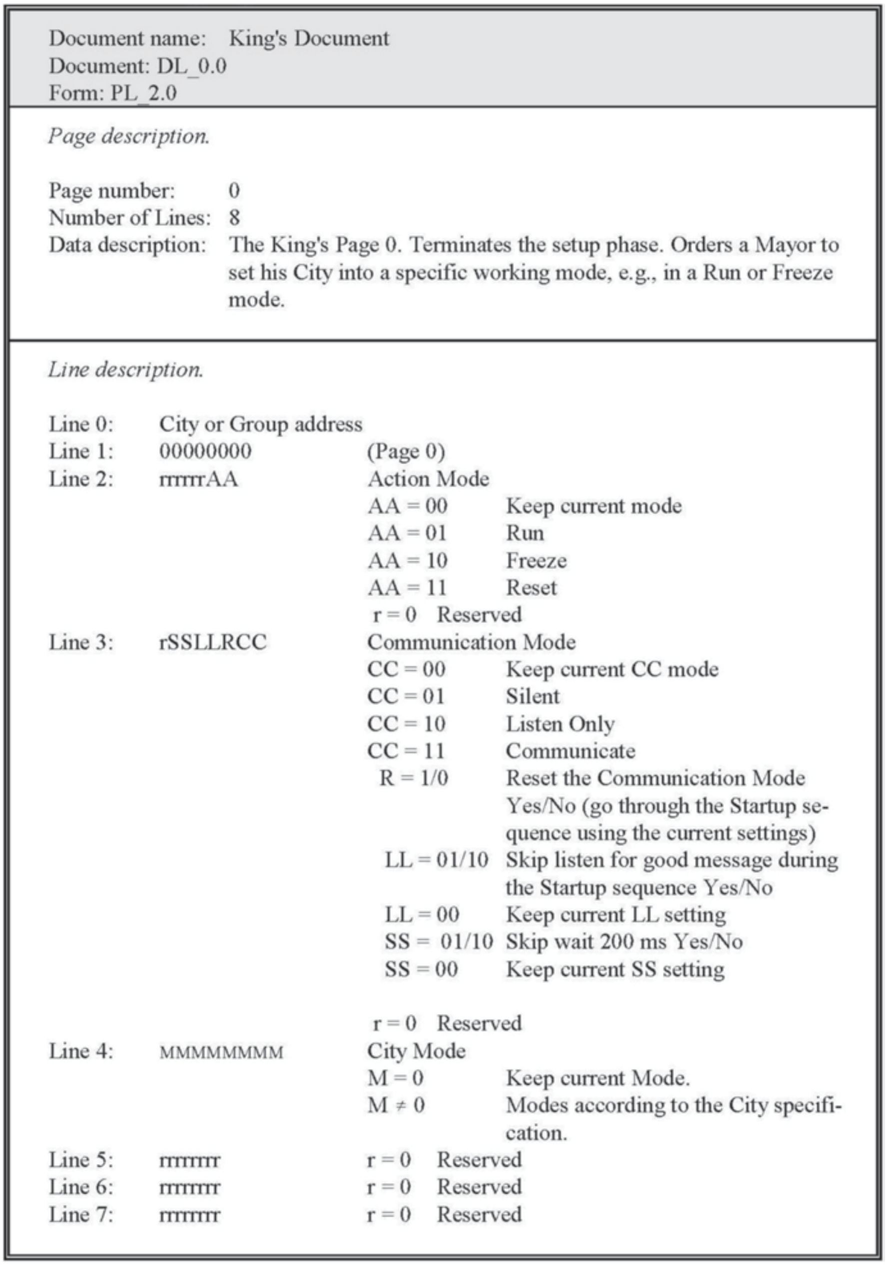
sary, but then each module has then to be updated before it is integrated into the system.) The first Line (first byte in the CAN data field) is a Group or City address. Any module in a system needs to have a physical address, a number between 1 and 255, before it is connected to the system. All Cities in the Kingdom belong to a Group with the address 0. This address can be used for broadcasting purposes. It is also possible to assign a City to additional Groups. The Group address is given by the King and can be any number between 1 and 255 not used as a City address. This feature can, for example, be used to freeze a part of the Kingdom in an emergency situation. The second Line is the King's Page number. Numbers 0–31 are reserved for the core CK rules, 32–127 for future additions needed to enable the integration of other HLPs. The numbers 128–255 can be used for City-specific needs. King's Page 0 is shown in Table 4.22.

4.5.5.1 Action Mode

It relates to actual City Mode and Communication Mode. The reset behaviour may be different in different Communication Modes regarding stored bit-timing register settings and stored parameter values. Action Modes have to be defined by the City Founder. It is recommended that the City supports at least the following Action Modes:

- Run: The City is functional and operating.
- Freeze: The City takes a safe state, still responding to King's Letters.
- Reset: The City performs a restart including the Start-up procedure in Sect. 9. The 200 ms at 125 Kbit/s sequence may be omitted.

Table 4.22 Form of the King’s Page 0



Document name: King's Document
Document: DL_0.0
Form: PL_2.0

Page description.

Page number:	0
Number of Lines:	8
Data description:	The King's Page 0. Terminates the setup phase. Orders a Mayor to set his City into a specific working mode, e.g., in a Run or Freeze mode.

Line description.

Line 0:	City or Group address		
Line 1:	00000000	(Page 0)	
Line 2:	rrrrrrAA	Action Mode	
		AA = 00	Keep current mode
		AA = 01	Run
		AA = 10	Freeze
		AA = 11	Reset
		r = 0	Reserved
Line 3:	rSSLLRCC	Communication Mode	
		CC = 00	Keep current CC mode
		CC = 01	Silent
		CC = 10	Listen Only
		CC = 11	Communicate
		R = 1/0	Reset the Communication Mode Yes/No (go through the Startup sequence using the current settings)
		LL = 01/10	Skip listen for good message during the Startup sequence Yes/No
		LL = 00	Keep current LL setting
		SS = 01/10	Skip wait 200 ms Yes/No
		SS = 00	Keep current SS setting
		r = 0	Reserved
Line 4:	MMMMMMMM	City Mode	
		M = 0	Keep current Mode.
		M ≠ 0	Modes according to the City specification.
Line 5:	rrrrrrrr	r = 0	Reserved
Line 6:	rrrrrrrr	r = 0	Reserved
Line 7:	rrrrrrrr	r = 0	Reserved

4.5.5.2 Communication Modes

- Silent: The Postmaster will be silent but still receives Letters and notifies the Mayor when Letters are accepted. The Postmaster will transmit neither acknowledgement bit nor error or overload frames.
- Listen Only: The Postmaster will be fully active but the Mayor will send Letters only upon the King's request.
- Communicate: Normal communication.

4.5.5.3 City Mode

City-specific modes, e.g. configuration mode, service mode, working mode, etc. For each City Mode the City Founder has to define how the City will work on Action Mode and Communication Mode commands from the King.

4.5.6 Mayor's Document

Each Mayor has a Document with some Pages and an Envelope of his own by which he can respond to the King. The King will broadcast a Base Number and the Mayor will use the Envelope that equals the sum of the City Address and the Base Number. The Mayor's Pages 0 and 1 are mandatory (Table 4.23 and 4.24):

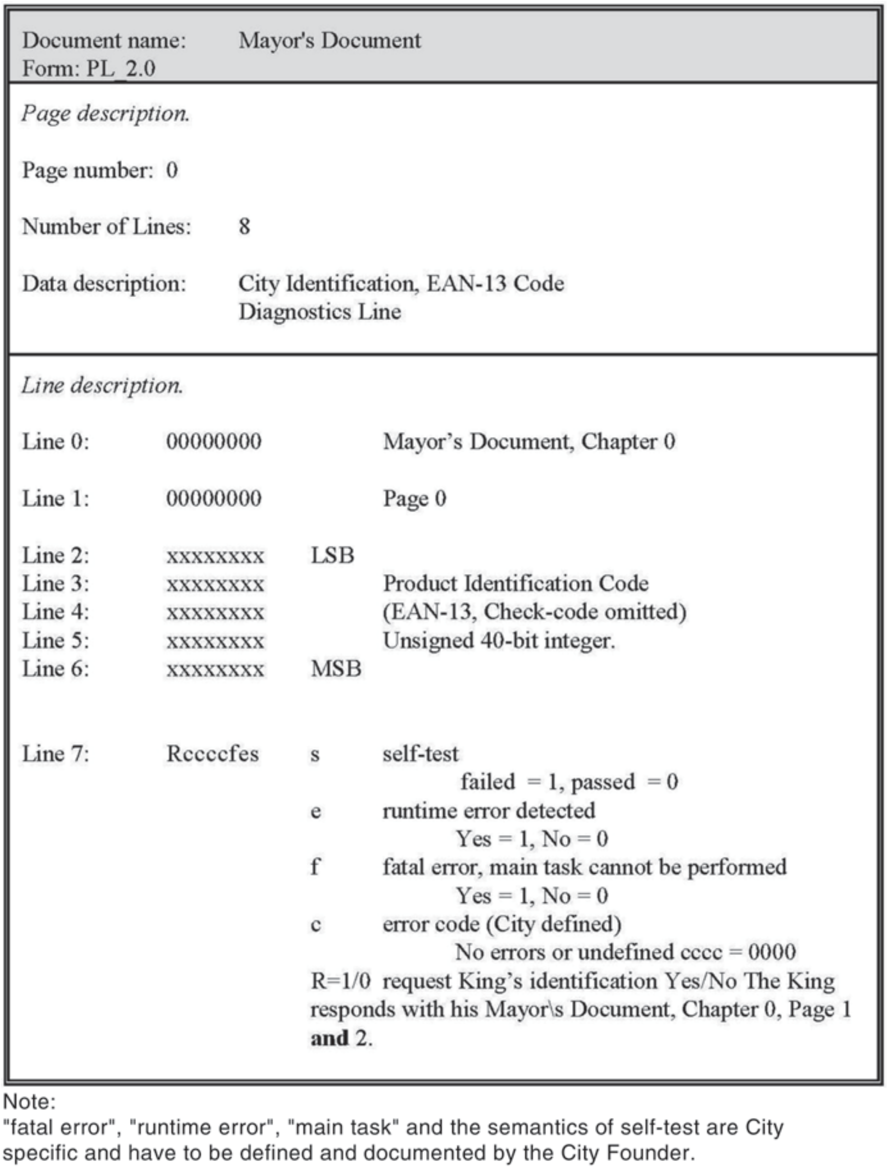
A City is fully identified by the Mayor's Page 1 and 2. Any King's Page can be mirrored by a Mayor's Page and the current setup of a City can be checked by asking for those Pages.

4.5.7 City Organization

The City Founder (the module designer) always knows what information his City must receive and what it can transmit in different situations but frequently he does not know how his City will be used in a specific Kingdom. It is only known by the Kingdom Founder (the system designer). A convenient way for the City Founder to escape the problem of how to receive and transmit information is to leave it to the Kingdom Founder. This is done by organizing the City information into Lists where selectable information blocks are referenced by records. Lists are enumerated from 0 to 253 and each list can hold up to 256 records. The King can use some King's Pages to order the Mayor to construct new Pages by referring to records in the Lists and place them in new Documents. In this way, modules can be optimized to the system requirements and the use of bandwidth optimized as only required data are transmitted. No profiles like the ones in CANopen and DeviceNet have to be defined.

A full-blown City may have all of the following Lists:

Table 4.23 Form for the Mayor's Page 0

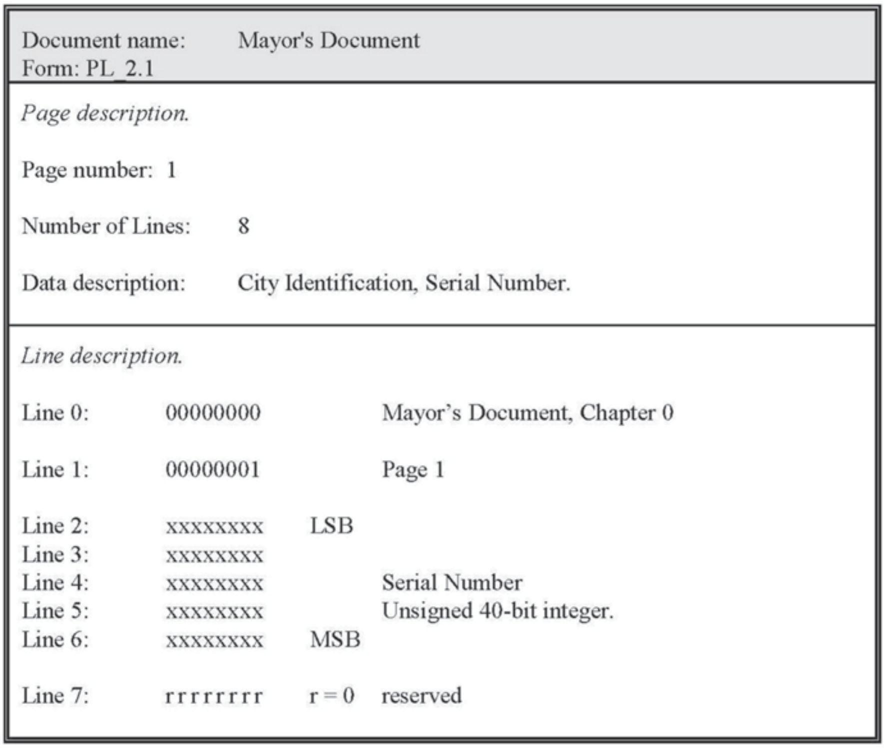


Document name: Form: PL_2.0	Mayor's Document		
Page description.			
Page number: 0			
Number of Lines:	8		
Data description:	City Identification, EAN-13 Code Diagnostics Line		
Line description.			
Line 0:	00000000		Mayor's Document, Chapter 0
Line 1:	00000000		Page 0
Line 2:	xxxxxxxx	LSB	
Line 3:	xxxxxxxx		Product Identification Code
Line 4:	xxxxxxxx		(EAN-13, Check-code omitted)
Line 5:	xxxxxxxx		Unsigned 40-bit integer.
Line 6:	xxxxxxxx	MSB	
Line 7:	Rccccfes	s	self-test failed = 1, passed = 0
		e	runtime error detected Yes = 1, No = 0
		f	fatal error, main task cannot be performed Yes = 1, No = 0
		c	error code (City defined) No errors or undefined cccc = 0000
		R=1/0	request King's identification Yes/No The King responds with his Mayor\s Document, Chapter 0, Page 1 **and** 2.

Note:
"fatal error", "runtime error", "main task" and the semantics of self-test are City specific and have to be defined and documented by the City Founder.

1. Document List: The King selects Documents for transmission and reception by placing Documents into Folders. Available Documents are listed in Document Lists. These Lists are of two different types: Receive or Transmit.
2. Page List: A City can offer the King the opportunity to construct Documents from predefined Page Forms. These Forms are then listed in one or more Page Lists.
3. Line List: A City can offer the King the opportunity to construct Pages by predefined Line Forms. These Forms are then listed in one or more Line Lists.

Table 4.24 Form for the Mayor's Page 1



Document name:	Mayor's Document		
Form: PL_2.1			
Page description.			
Page number: 1			
Number of Lines:	8		
Data description:	City Identification, Serial Number.		
Line description.			
Line 0:	00000000		Mayor's Document, Chapter 0
Line 1:	00000001		Page 1
Line 2:	xxxxxxxx	LSB	
Line 3:	xxxxxxxx		
Line 4:	xxxxxxxx		Serial Number
Line 5:	xxxxxxxx		Unsigned 40-bit integer.
Line 6:	xxxxxxxx	MSB	
Line 7:	rrrrrrrr	r = 0	reserved

4. Dot List: A City can offer the King the opportunity to construct Lines by predefined Dot Forms. These Forms are then listed in one or more Line Lists.
5. Item List: In a Compressed Page, data can be placed not only in the CAN data field but also in the CAN ID field. A City supporting Compressed Pages or Letters has an Item List where available constants, parameters, variables, etc. and information about them can be found. By using references to List and Record numbers, the King can instruct the Mayor where to place or read specific data anywhere he likes in a CAN message. Data can then be extended into the CAN identifier or integrated as a part of the identifier field.

4.5.8 The Folder

A Folder is the link between a Document and the Postal System. One or more Envelopes are assigned to a Folder that contains one Document. A Folder Label contains all necessary Postal information for the exchanging of Letters between Cities.

A City can have up to 256 different Folders for incoming or outgoing Documents. The Mayor puts the Documents for the information that he will send or receive into these Folders. The King will then assign Envelopes to Folders containing a Document of interest for the Kingdom. A Folder can be fixed, i.e. the Document in the Folder is predefined or dynamic, i.e. the King can order the Mayor to put a Document into a given Folder. The advantages of letting the King to decide which Documents will be put into which Folders are that matching Documents in different Cities will have a common identification throughout the Kingdom. The disadvantage is that this requires some software and a City Founder may find it too expensive and choose to have the Documents placed in fixed Folders to save memory.

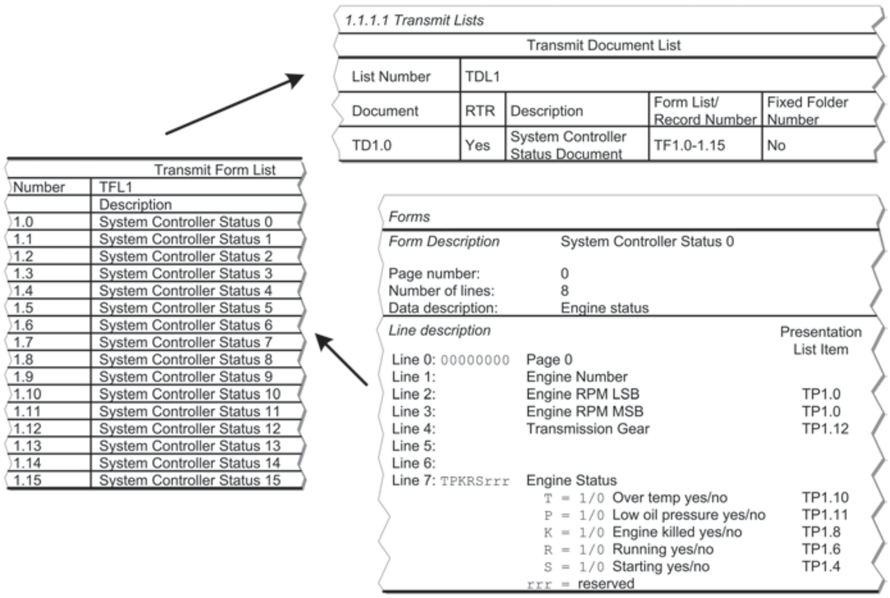
4.5.9 Folder Label

A Folder always has a label, the Folder Label. It contains the following information:

- Folder Number.
- Document List Number.
- Document Number.
- Transmit/Receive mark.
- The CAN Control Field according to the CAN specification.
- Remote Envelope(s).
- An Envelope can be set as "remote" according to the CAN specification by the RTR bit. How RTR set to 1 is interpreted is dependent on the application corresponding to the Document in the Folder and has to be defined in the City documentation.
- Enable/disable the Folder.
- By disabling the Folder, any CAN communication by the application corresponding to in this Folder is interrupted.
- Envelope(s) assigned to this Folder.
- Envelope(s) enable/disable.
- The use of an Envelope can be switched on or off with an enable/disable tag.

How a Page is identified in a Transmit Form List and put into a transmit Document is depicted in Fig. 4.34. Receiving Cities use the same Page Form in their corresponding Receive Document.

As shown, a City Founder (module designer) does not have to care about how his module will exchange information with other modules in a system. He has defined what information his module needs and the timing constrictions. He has also specified what kind of information the module can make available to other modules. A Kingdom Founder can later on adapt the module to the needs of his system by transmitting King's Pages at a start-up procedure and even dynamically tune it during run time. Thus, the system designer can adapt and control the system using the King in the Capital. Some diagnostics can only be made at the system level by monitoring the traffic and getting internal information from modules. Any internal



1.1.1.1 Transmit Lists

Transmit Document List

List Number	TDL1			
Document	RTR	Description	Form List/ Record Number	Fixed Folder Number
TD1.0	Yes	System Controller Status Document	TF1.0-1.15	No

Transmit Form List

Number	TFL1
	Description
1.0	System Controller Status 0
1.1	System Controller Status 1
1.2	System Controller Status 2
1.3	System Controller Status 3
1.4	System Controller Status 4
1.5	System Controller Status 5
1.6	System Controller Status 6
1.7	System Controller Status 7
1.8	System Controller Status 8
1.9	System Controller Status 9
1.10	System Controller Status 10
1.11	System Controller Status 11
1.12	System Controller Status 12
1.13	System Controller Status 13
1.14	System Controller Status 14
1.15	System Controller Status 15

Forms

Form Description	System Controller Status 0
Page number:	0
Number of lines:	8
Data description:	Engine status

Line description			Presentation List Item
Line 0:	00000000	Page 0	
Line 1:		Engine Number	
Line 2:		Engine RPM LSB	TP1.0
Line 3:		Engine RPM MSB	TP1.0
Line 4:		Transmission Gear	TP1.12
Line 5:			
Line 6:			
Line 7:	TPKRSrrr	Engine Status	
	T = 1/0	Over temp yes/no	TP1.10
	P = 1/0	Low oil pressure yes/no	TP1.11
	K = 1/0	Engine killed yes/no	TP1.8
	R = 1/0	Running yes/no	TP1.6
	S = 1/0	Starting yes/no	TP1.4
	rrr =	reserved	

Fig. 4.34 Identification of a sending system in a transmission document list

status and parameter value in a module can be made available to the King in the Capital. Consequently, any decision on what to do in specific situations can be left to the system designer to decide among a set of alternatives.

A small CK system is depicted in Fig. 4.35. City 1 measures oil temperature and City 2 measures water temperature. Each City only measures temperature and makes the measured values available to the system. They do not have to know what they are measuring and when to transmit. City 3 receives both temperature values and has to distinguish between the two. This is easily done by assigning an Envelope to the Temperature Transmit Document in City 1 and the same Envelope to the oil temperature Receive Document in City 3. The temperature in City 2 is connected to the water temperature in City 3. With CK, no profiles are necessary as the King can set up a City to match the needs of the system. The module designer has great freedom to integrate a variety of options to meet the requirements of different HLPs and profiles. The module can then be adapted to a specific HLP and profile using proper King's Pages at an end of line programming. A system designer can also integrate modules made for other HLPs into his system by adapting other modules to the runtime behaviour of the integrated ones. Any HLP-specific start-up procedure, e.g. the "Duplicate MAC ID check" in DeviceNet or "Address Claim" in J1939 can be simulated by the King to make the integrated module happy.

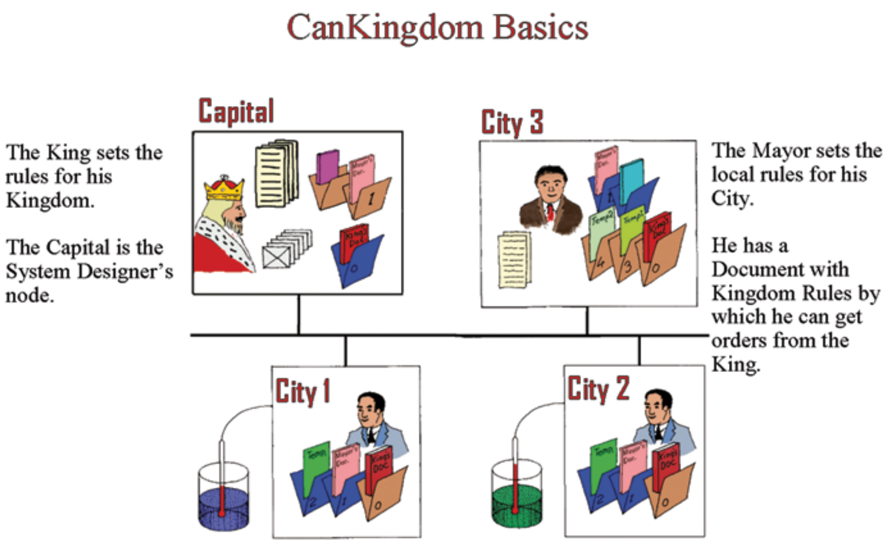

CanKingdom Basics
Capital
City 3
The King sets the rules for his Kingdom.
The Capital is the System Designer's node.
The Mayor sets the local rules for his City.
He has a Document with Kingdom Rules by which he can get orders from the King.
City 1
City 2

Fig. 4.35 CanKingdom basics

4.5.10 *Composability and Membership*

The composability in CK relies on the King, i.e. the system needs to have a Capital (a system node) and the start-up procedure. During the start-up procedure, all connected Cities are identified. Any new City can be properly configured. For a safe cold start, the following are necessary:

- Each module conducts a self-test.
- Each node connects to the CAN network in silent mode, i.e. without transmitting ACK bits, and listens for a specific message at 125 kbps for 2 s.
- Switch to stored bit rate for the current system.
- When a valid message is received, the module switches to listen-only mode, i.e. it participates in the CAN error checking and acknowledgement, but does not transmit and waits for the King's Page 1 with the Base Number.
- The King sends King's Page 1, either with the Group Address 0 or with individual addresses, to make every module respond with its European Article Number (EAN) and serial number (Fig. 4.36).

In this way, the King has complete control of the system. The King can check that all anticipated modules are connected and working correctly before the system is turned into runtime mode. During runtime mode, the King can supervise the system and check for different types of errors as missed schedules, unauthorized bus traffic, mismatching values, etc. The King can also act as a gateway to external tools or systems and provide full documentation of the settings of each node and their respective states.

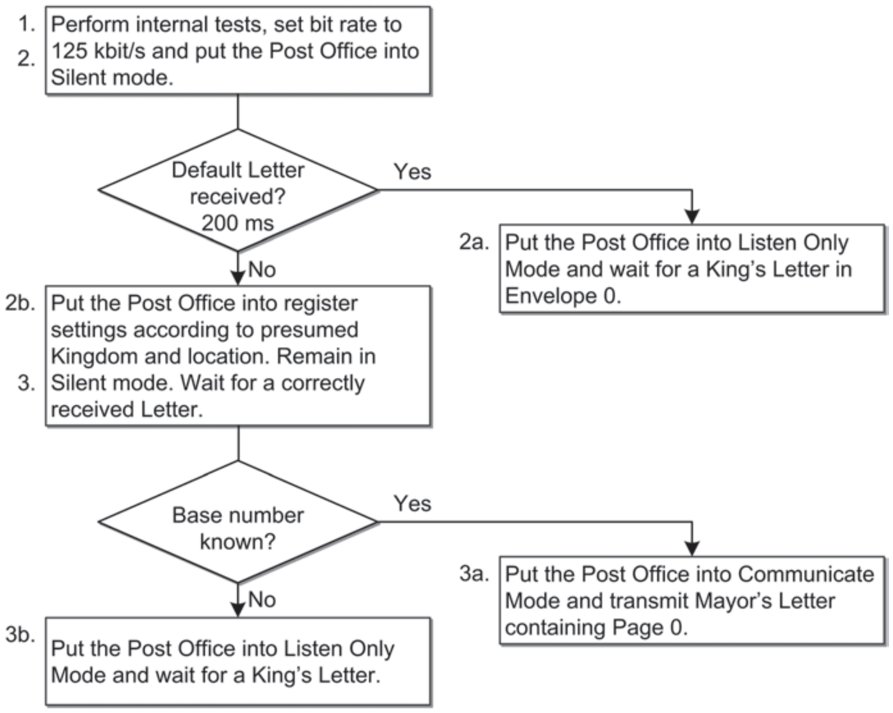

1.
2.
Perform internal tests, set bit rate to 125 kbit/s and put the Post Office into Silent mode.
Default Letter received? 200 ms
Yes
2a. Put the Post Office into Listen Only Mode and wait for a King's Letter in Envelope 0.
No
2b.
3.
Put the Post Office into register settings according to presumed Kingdom and location. Remain in Silent mode. Wait for a correctly received Letter.
Base number known?
Yes
3a. Put the Post Office into Communicate Mode and transmit Mayor's Letter containing Page 0.
No
3b. Put the Post Office into Listen Only Mode and wait for a King's Letter.

Fig. 4.36 CanKingdom setting

If any module is set to a false bit rate, it will not disturb the network as it stays in silent mode. An external setup tool (or the King) will always be able to connect at 125 kbit/s and correct the bit-timing register settings.