BRITISH STANDARD

BS EN 50325-4:2002

Industrial communications subsystem based on ISO 11898 (CAN) for controller-device interfaces —

Part 4: CANopen

The European Standard EN 50325-4:2002 has the status of a British Standard

ICS 43.180

British Standards

National foreword

This British Standard is the official English language version of EN 50325-4:2002.

The UK participation in its preparation was entrusted to Technical Committee AMT/7, Industrial communications: Process measurement and control, including Fieldbus, which has the responsibility to:

— aid enquirers to understand the text;

— present to the responsible international/European committee any enquiries on the interpretation, or proposals for change, and keep the UK interests informed;

— monitor related international and European developments and promulgate them in the UK.

A list of organizations represented on this committee can be obtained on request to its secretary.

Cross-references

The British Standards which implement international or European publications referred to in this document may be found in the *BSI Catalogue* under the section entitled "International Standards Correspondence Index", or by using the "Search" facility of the *BSI Electronic Catalogue* or of British Standards Online.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 3 February 2003

Summary of pages

This document comprises a front cover, an inside front cover, the EN title page, pages 2 to 115 and a back cover.

The BSI copyright date displayed in this document indicates when the document was last issued.

Amendments issued since publication

Amd. No.	Date	Comments

ISBN 0 580 41175 3

EUROPEAN STANDARD **EN 50325-4**

NORME EUROPÉENNE

EUROPÄISCHE NORM December 2002

ICS 43.180

English version

Industrial communications subsystem based on ISO 11898 (CAN) for controller-device interfaces Part 4: CANopen

Sous-système de communications industriel basé sur l'ISO 11898 (CAN) pour les interfaces des dispositifs de commande
Partie 4: CANopen

Industrielles Kommunikationssubsystem basierend auf ISO 11898 (CAN)
Teil 4: CANopen

This European Standard was approved by CENELEC on 2002-07-01. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the Central Secretariat or to any CENELEC member.

This European Standard exists in one official version (English). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the Central Secretariat has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Czech Republic, Denmark, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Luxembourg, Malta, Netherlands, Norway, Portugal, Slovakia, Spain, Sweden, Switzerland and United Kingdom.

CENELEC
European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Central Secretariat: rue de Stassart 35, B - 1050 Brussels

Foreword

This European Standard was prepared by the Technical Committee CENELEC TC 65CX, Fieldbus.

The text of the draft was submitted to the Unique Acceptance Procedure and was approved by CENELEC as EN 50325-4 on 2002-07-01.

The following dates were fixed:

- latest date by which the EN has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2003-07-01
- latest date by which the national standards conflicting with the EN have to be withdrawn	(dow)	2005-07-01

Annexes designated "normative" are part of the body of the standard.

Annexes designated "informative" are given for information only.

In this standard, annexes A and B are normative and annexes C, D and E are informative.

This European standard is part of EN 50325 which consists of four parts:

Part 1	General requirements
Part 2	DeviceNet
Part 3	Smart Distributed System (SDS)
Part 4	CANopen

The specifications for DeviceNet,SDS and CANopen are based on ISO 11898 *Controller area network (CAN) for high-speed communication*, a broadcast-oriented communications protocol. However, ISO 11898 specifies only part of a complete communication system, and additional specifications are needed for other layers to ensure precise data exchange functionality and support of inter-operating devices.

General information on licensing and patents

Attention is drawn to the possibility that some of the elements of the European Standard EN 50325-4 may be the subject of patent rights. CENELEC shall not be held responsible for identifying any or all such patent rights

If during the application of those Standards Intellectual Property Rights may appear and will not be made available on reasonable and non discriminatory terms and conditions to anyone wishing to obtain such a license, applying the rules of CEN/CENELEC Memorandum 8, this fact shall be brought to the attention of CENELEC Central Secretariat for further action.

Contents

Introduction

CANopen is intended for use in, but is not limited to, industrial automation applications. These applications may include devices such as generic digital and analogue input/output modules, motion controllers, human machine interfaces, sensors, closed-loop controllers, encoders, hydraulic valves, and programmable controllers.

1 Scope

EN 50325-4 specifies the following particular requirements for CANopen:

- requirements for interfaces between programmable controllers and devices with input/output capabilities;
- normal service conditions for devices;
- constructional and performance requirements.

2 Normative references

EN 50081-2	1993	Electromagnetic compatibility (EMC) – Generic emission standard - Part 2: Industrial environment
EN 61000-6-2	1999	Electromagnetic compatibility (EMC) - Generic standards - Part 2: Immunity for industrial environments
EN 55011	1998	Industrial, scientific and medical (ISM) radio-frequency equipment – Radio disturbance characteristics - Limits and methods of measurement (CISPR 11: 1997, mod.)
EN 61000-4		Electromagnetic compatibility (EMC) – Part 4: Testing and measurement techniques
EN 61131-3	1993	Programmable controllers – Part 3: Programming languages (IEC 61131-3:1993)
ISO 11898	1993	Road vehicles - Interchange of digital information - Controller area network (CAN) for high-speed communication
ISO 646	1991	Information technology - ISO 7-bit coded character set for information interchange
ISO 7498-1	1994	Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model
ISO 8859	1998	Information technology - 8-bit single-byte coded graphic character sets

3 Definitions

For the purpose of EN 50325-4 the definitions of EN 50325-1 and the following apply.

3.1
Automatic Repeat Request (ARQ)
scheme used to confirm the transmission of an SDO block

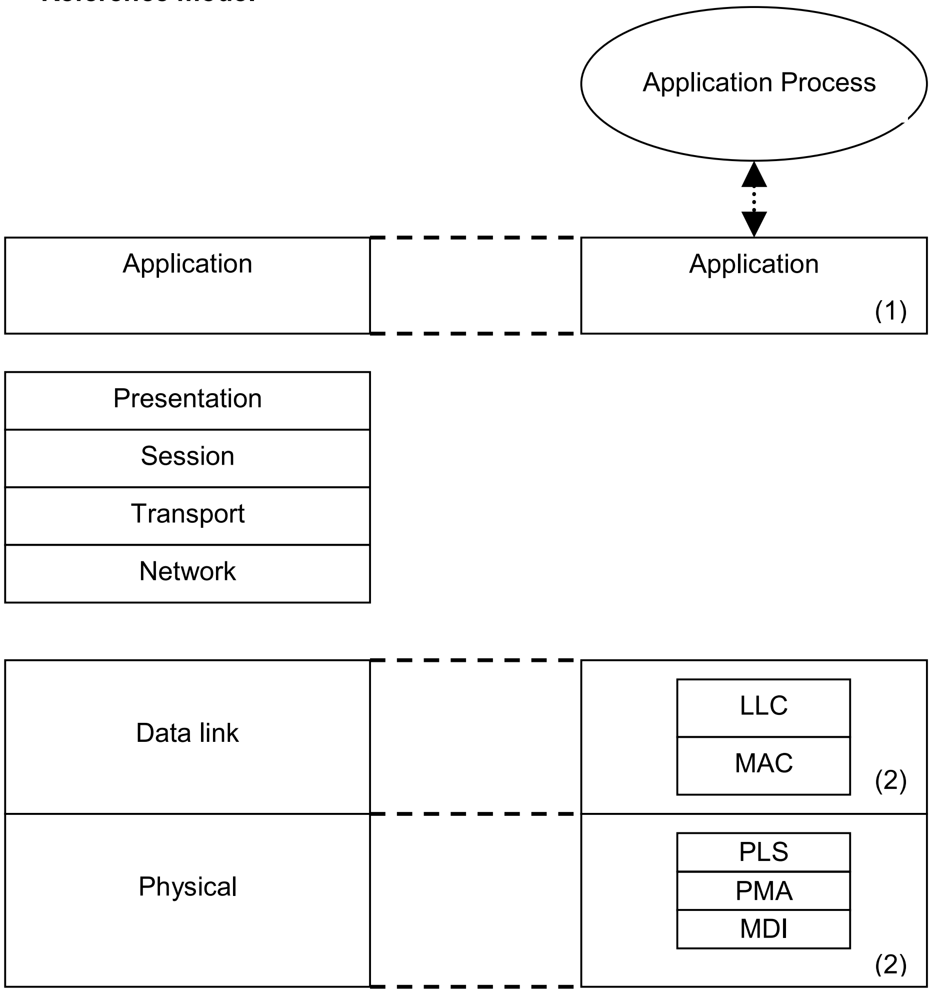
3.2
Node-ID
number, uniquely assigned to an NMT (NetworkManagemenT) slave; if 0, the NMT protocols address all NMT slaves

4 Classifications

4.1 General

Networks compliant to EN 50235-4 shall use the following reference model, device model, and communication model.

4.2 Reference model



(1) specified in this European Standard

(2) specified in ISO 11898

Figure 1 – Comparison with OSI reference model

The communication concept is described similar to the OSI reference model (left side of Figure 1).

4.2.1 Application layer:

The application layer comprises a concept to configure and communicate real-time-data as well as the mechanisms for synchronisation between devices. The functionality the application layer offers to an application is logically divided over different *service objects* in the application layer. A service object offers a specific functionality and all the related services. These services are described in the *Service Specification* of that service object.

Applications interact by invoking services of a service object in the application layer. To realise these services, this object exchanges data via the CAN network with (a) peer service object(s) via a protocol. This protocol is described in the *Protocol Specification* of that service object.

4.2.2 Service primitives:

Service primitives are the means by which the application and the application layer interact. There are four different primitives

- a *request* is issued by the application to the application layer to request a service,
- an *indication* is issued by the application layer to the application to report an internal event detected by the application layer or indicate that a service is requested,
- a *response* is issued by the application to the application layer to respond to a previous received indication,
- a *confirmation* is issued by the application layer to the application to report the result of a previously issued request.

4.2.3 Application layer service types

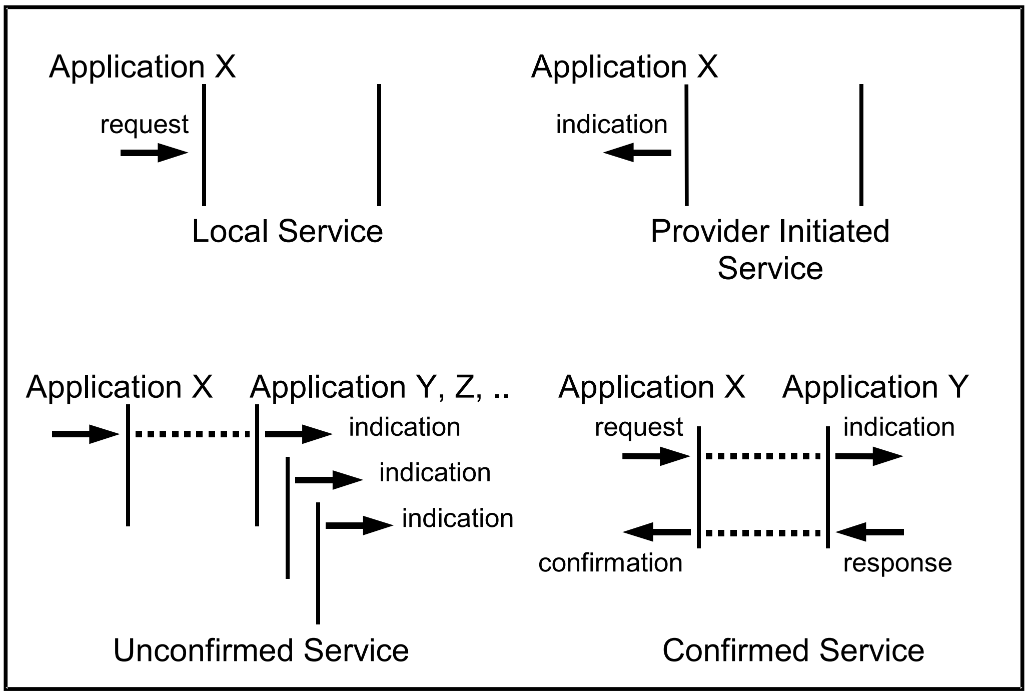


Figure 2 - Service types

A *service type* defines the primitives that are exchanged between the application layer and the co-operating applications for a particular service of a service object (see Figure 2).

- A *local service* involves only the local service object. The application issues a request to its local service object that executes the requested service without communicating with (a) peer service object(s).
- An *unconfirmed service* involves one or more peer service objects. The application issues a request to its local service object. This request is transferred to the peer service object(s). Each of them pass it to their application as an indication. The result is not confirmed back.
- A *confirmed service* may involve only one peer service object. The application issues a request to its local service object. This request is transferred to the peer service object that passes it to the other application as an indication. The other application issues a response that is transferred to the originating service object that passes it as a confirmation to the requesting application.
- A *provider initiated service* involves only the local service object. The service object (being the service provider) detects an event not solicited by a requested service. This event is then indicated to the application.

Unconfirmed and confirmed services are collectively called *remote services*.

4.3 Device model

4.3.1 General

A device is modelled as follows (see Figure 3):

- communication – This function unit provides the communication objects and the appropriate functionality to transport data items via the underlying network structure;
- object dictionary – The Object dictionary is a collection of all the data items which have an influence on the behaviour of the application objects, the communication objects and the state machine used on this device;
- application – The application comprises the functionality of the device with respect to the interaction with the process environment.

Thus the Object dictionary serves as an interface between the communication and the application. The complete description of a device's application with respect to the data items in the Object dictionary is named *device profile*.

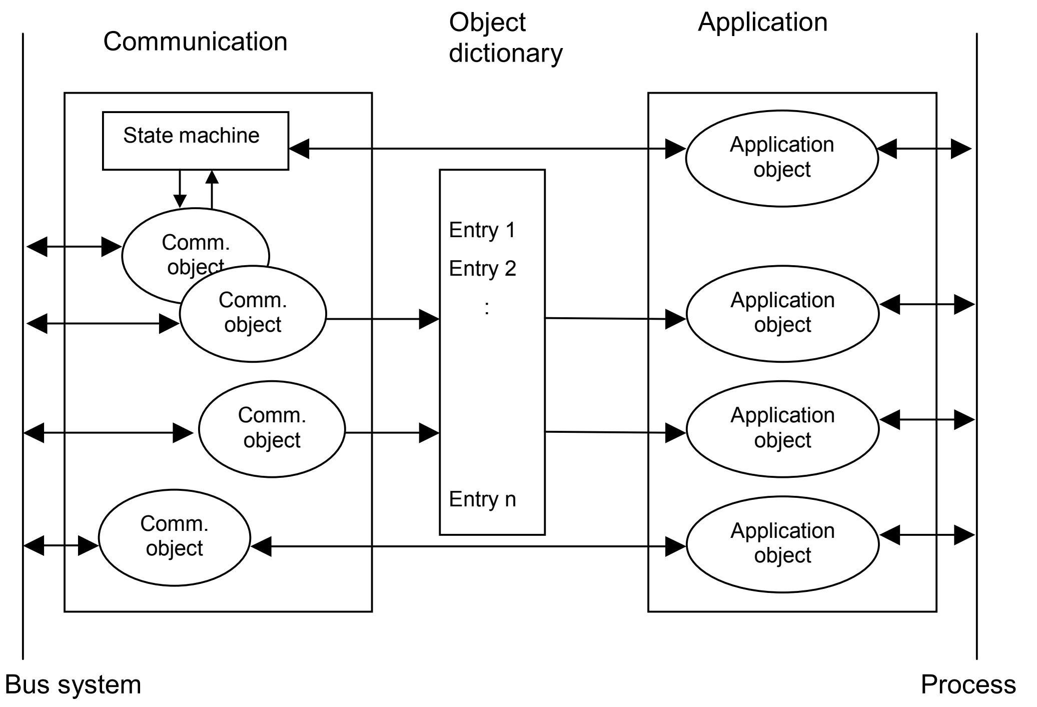


Figure 3 - Device model

4.3.2 Object dictionary

4.3.2.1 General

The most important part of a device profile is the Object dictionary description. The Object dictionary is essentially a grouping of objects accessible via the network in an ordered pre-defined fashion. Each object within the dictionary shall be addressed using a 16-bit index.

The overall layout of the standard Object dictionary is shown in Table 1. This layout closely conforms to other industrial serial bus system concepts.

Table 1 - Object dictionary structure

Index (hex)	Object
0000	Not used
0001-001F	Static data types
0020-003F	Complex data types
0040-005F	Manufacturer specific complex data types
0060-007F	Device profile specific static data types
0080-009F	Device profile specific complex data types
00A0-0FFF	Reserved for further use
1000-1FFF	Communication profile area
2000-5FFF	Manufacturer specific profile area
6000-9FFF	Standardised device profile area
A000-FFFF	Reserved for further use

The object dictionary may contain a maximum of 65536 entries which are addressed through a 16-bit index.

The Static Data Types at indices 0001h through 001Fh shall contain type definitions for simple data types like boolean, integer, floating point, string, etc. These entries are included for reference only, they shall not be read or written.

Complex Data Types at indices 0020h through 003Fh are pre-defined structures that are composed of simple data types and are common to all devices.

Manufacturer Specific Complex Data Types at indices 0040h through 005Fh are structures composed of standard data types but are specific to a particular device.

Device profiles may define additional data types specific to their device type. The static data types defined by the relevant device profile are listed at indices 0060h - 007Fh, the complex ones at indices 0080h - 009Fh.

A device may provide the structure of the supported complex data types (indices 0020h - 005Fh and 0080h - 009Fh) at read access to the corresponding index. Sub-index 0 then provides the number of entries at this index, and the following sub-indices contain the data type encoded as UNSIGNED16 according to Table B.4.

The Communication Profile Area at indices 1000h through 1FFFh shall contain the communication specific parameters for the CAN network. These entries shall be common to all devices.

The standardised device profile area at indices 6000h through 9FFFh contains all data objects common to a class of devices that may be read or written via the network. The device profiles may use entries from 6000h to 9FFFh to describe the device parameters and the device functionality. Within this range up to 8 different devices may be described. In such a case the devices are denominated Multiple Device Modules. Multiple Device Modules are composed of up to 8 device profile segments. By this feature it is possible to build devices with multiple functionality.

The object entries 6000h to 67FFh of each additional device profiles shall be shifted as follows:

-	6000h to 67FFh	for	device 0
-	6800h to 6FFFh	for	device 1
-	7000h to 77FFh	for	device 2
-	7800h to 7FFFh	for	device 3
-	8000h to 87FFh	for	device 4
-	8800h to 8FFFh	for	device 5
-	9000h to 97FFh	for	device 6
-	9800h to 9FFFh	for	device 7

The PDO distribution shall be used for every segment of a Multiple Device Module with an offset of 64, e.g. the first PDO of the second segment gets the number 65. In this way a system with a maximum of 8 segments is supported.

The object dictionary concept caters for optional device features which means a manufacturer does not have to provide certain extended functionality on his devices. However, if he implements this optional functionality, he shall be compliant to definitions given in this European Standard.

Space is left in the Object dictionary at indices 2000h through 5FFFh, which may be used for truly manufacturer-specific functionality.

4.3.2.2 Index and sub-Index usage

A 16-bit index shall be used to address all entries within the Object dictionary. In case of a simple variable the index references the value of this variable directly. In case of records and arrays, however, the index addresses the whole data structure.

To allow individual elements of structures of data to be accessed via the network a sub-index is defined. For single Object dictionary entries such as an UNSIGNED8, BOOLEAN, INTEGER32, etc. the value for the sub-index shall be always zero. For complex Object dictionary entries such as arrays or records with multiple data fields the sub-index references fields within a data-structure pointed to by the main index. The fields accessed by the sub-index may be of different data types.

4.4 Communication model

4.4.1 General

The communication model specifies the different communication objects and services and the available triggering modes of message transmission.

The communication model supports the transmission of synchronous and asynchronous messages. By means of synchronous message transmission a network wide co-ordinated data acquisition and data actuation is possible. The synchronous transmission of messages is supported by pre-defined communication objects (Sync message, time stamp message). Synchronous messages are transmitted with respect to the pre-defined Sync message, asynchronous messages may be transmitted at any time.

Due to the event-driven character of the underlying communication mechanism it is possible to define inhibit times for the communication. In order to guarantee that even communication objects with low priorities are transmitted in an appropriate time, communication objects may be assigned an inhibit-time. The inhibit-time of a communication object defines the minimum time that shall elapse between two consecutive invocations of a transmission service for that communication object. Inhibit-times may be assigned by the application.

With respect to their functionality, three types of communication relationships are distinguished

- master/slave relationship (Figure 4 and Figure 5),
- client/server relationship (Figure 6),
- producer/consumer relationship (Figure 7 and Figure 8).

4.4.2 Master/slave relationship

At any time there is exactly one device in the network serving as a master for a specific functionality. All other devices in the network are considered as slaves. The master issues a request and the addressed slave(s) respond(s) if the protocol requires this behaviour.

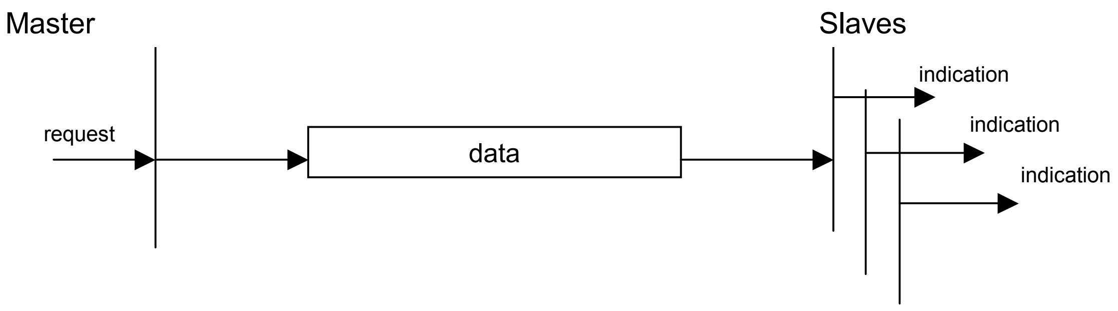


Figure 4 - Unconfirmed master/slave communication

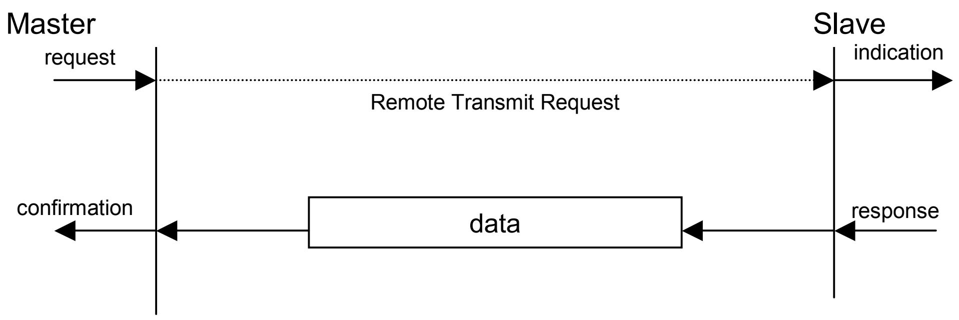


Figure 5 - Confirmed master/slave communication

4.4.3 Client/server relationship

This is a relationship between a single client and a single server. A client issues a request (upload/download) thus triggering the server to perform a certain task. After finishing the task the server answers the request.

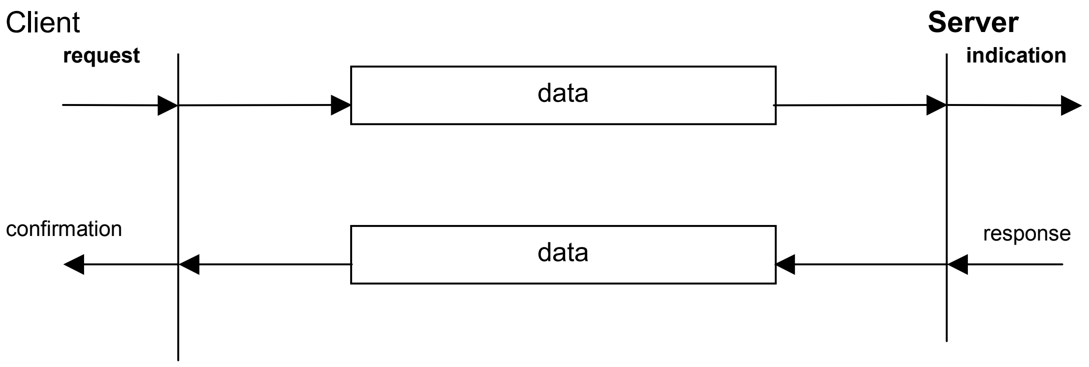


Figure 6 - Client/server communication

4.4.4 Producer/consumer relationship - Pull/push model

The producer/consumer relationship model involves a producer and zero or more consumer(s). The push model is characterised by an unconfirmed service requested by the producer. The pull model is characterised by a confirmed service requested by the consumer.

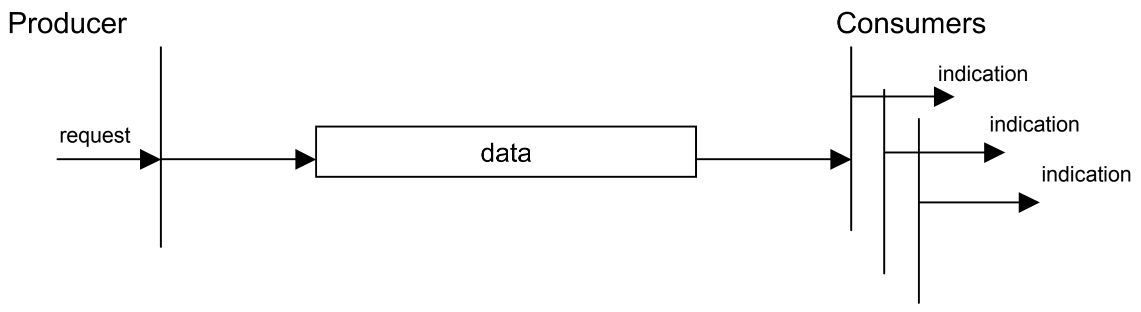


Figure 7 - Push model

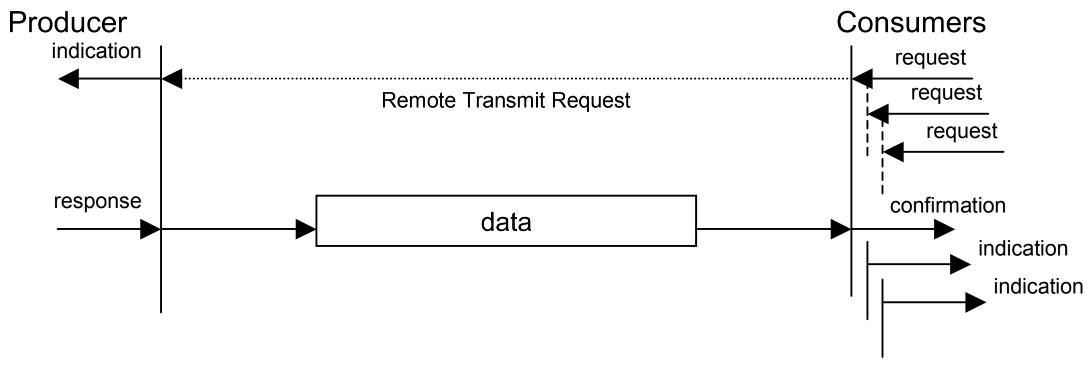


Figure 8 - Pull model

4.5 Data link layer

4.5.1 General

Networks compliant to EN 50235-4 shall be implemented on a data link layer and its sub-layers according to ISO 11898.

4.5.2 CAN frame type

This specification shall be implemented using CAN standard frames with 11-bit identifier field. It is not required to support the CAN extended frame with 29-bit identifier field.

However, as certain applications may require the usage of the extended frame with 29-bit identifier field the network may be operated in this mode, if all nodes support it.

5 Characteristics

5.1 Communication objects

5.1.1 General

The services and protocols describe the communication objects.

All services are described in a tabular form that contains the parameters of each service primitive that is defined for that service.

The primitives that are defined for a particular service determine the service type (e.g. unconfirmed, confirmed, etc.). How to interpret the tabular form and what service types exist is defined in 4.4 (Communication model).

All services assume that no failures occur in the data link and physical layer of the CAN network. These failures are resolved by the application and fall not in the scope of this European Standard.

5.1.2 Process Data Object (PDO)

5.1.2.1 General

The real-time data transfer is performed by means of Process Data Objects (PDO). The transfer of PDOs is performed with no protocol overhead.

The PDOs correspond to entries in the device Object dictionary and provide the interface to the application objects. Data type and mapping of application objects into a PDO is determined by a corresponding default PDO mapping structure within the Device Object dictionary.

If variable PDO-mapping is supported the number of PDOs and the mapping of application objects into a PDO may be transmitted to a device during the device configuration process (see Initialisation Procedure) by applying the SDO services to the corresponding entries of the Object dictionary.

Number and length of PDOs of a device is application specific and shall be specified within the device profile.

There are two kinds of use for PDOs. The first is data transmission and the second data reception. It is distinguished in Transmit-PDOs (TPDOs) and Receive-PDOs (RPDOs). Devices supporting TPDOs are PDO producers and devices which are able to receive PDOs are called PDO consumer. The PDO communication parameter (20h) and the PDO mapping parameter (21h) describe PDOs. The structure of these data types are explained in annex A. The PDO communication parameter describes the communication capabilities of the PDO. The PDO mapping parameter contains information about the contents of the PDOs (device variables). The indices of the corresponding Object dictionary entries shall be computed by the following formulas:

- RPDO communication parameter index = 1400h + RPDO-number –1;
- TPDO communication parameter index = 1800h + TPDO-number –1;
- RPDO mapping parameter index = 1600h + RPDO-number –1;
- TPDO mapping parameter index = 1A00h + TPDO-number –1.

For each PDO the pair of communication and mapping parameter shall be implemented. The entries mentioned above are described in annex B.

5.1.2.2 Transmission modes

The following PDO transmission modes are distinguished:

- Synchronous transmission
- Asynchronous transmission

In order to synchronise devices a synchronisation object (SYNC object) is transmitted periodically by a synchronisation application. The SYNC object is represented by a pre-defined communication object (see 5.1.4). In Figure 9 the principle of synchronous and asynchronous transmission is shown. Synchronous PDOs are transmitted within a pre-defined time-window immediately after the SYNC object. The principle of synchronous transmission is described in more detail in 5.2.

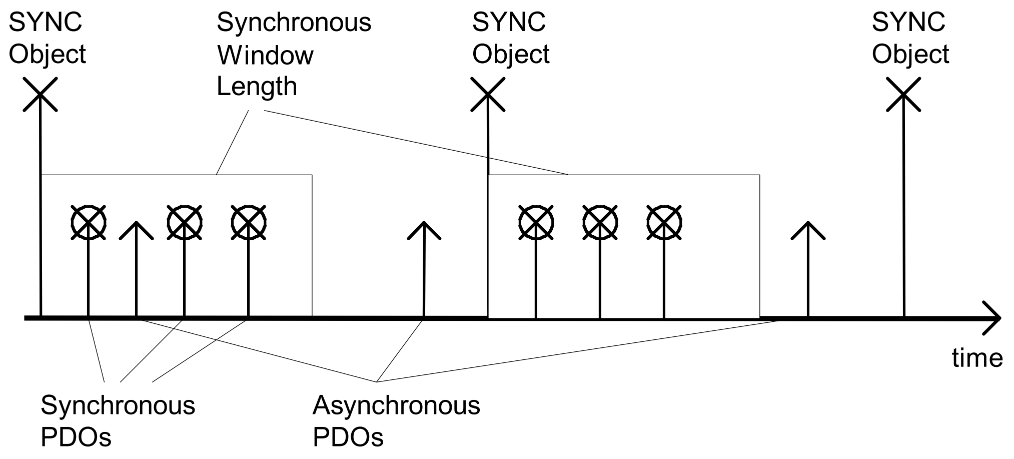


Figure 9 - Synchronous and asynchronous transmission

The transmission type parameter of a PDO specifies the transmission mode as well as the triggering mode.

For synchronous TPDOs the transmission type also specifies the transmission rate in form of a factor based on the basic SYNC-object transmission period. A transmission type of 0 means that the message shall be transmitted after occurrence of the SYNC but acyclic (not periodically), this is if an event occurred before the SYNC. A transmission type of 1 means that the message shall be transmitted with every SYNC object. A transmission type of n means that the message shall transmitted with every n-th SYNC object. Asynchronous TPDOs are transmitted without any relation to a SYNC.

The data of synchronous RPDOs received after the occurrence of a SYNC is passed to the application with the occurrence of the following SYNC, independent of the transmission rate specified by the transmission type. The data of asynchronous RPDOs is passed directly to the application.

5.1.2.3 Triggering modes

Three message triggering modes are distinguished:

- **Event driven**
  Message transmission is triggered by the occurrence of an object specific event. For synchronous PDOs this is the expiration of the specified transmission period, synchronised by the reception of the SYNC object.
  For acyclically transmitted synchronous PDOs and asynchronous PDOs the triggering of a message transmission is a device-specific event specified in the device profile.
- **Timer driven**
  Message transmission is either triggered by the occurrence of a device-specific event or if a specified time has elapsed without occurrence of an event.
- **Remotely requested**
  Transmission of an asynchronous PDO is initiated on receipt of a remote request initiated by any other device (PDO consumer).

5.1.2.4 PDO services

5.1.2.4.1 General

PDO transmission shall follow the producer/consumer relationship as described in 4.4.4.

Attributes:

- PDO number:	PDO number [1..512] for every user type on the local device
- user type:	one of the values {consumer, producer}
- data type:	according to the PDO mapping
- inhibit-time:	n*100 µs, n >> 0

5.1.2.4.2 Write PDO

For the Write PDO service the push model is valid. There are zero or more consumers of a PDO. A PDO shall have exactly one producer.

Through this service the producer of a PDO sends the data of the mapped application objects to the consumer(s) (see Table 2).

Table 2 - Write PDO

Parameter	*Request / Indication*
Argument	**Mandatory**[1]
PDO Number	mandatory
Data	mandatory
[1] *Mandatory* attributes shall be implemented	

5.1.2.4.3 Read PDO

For the Read PDO service the pull model is valid. There are one or more consumers of a PDO. A PDO shall have exactly one producer.

Through this service the consumer of a PDO requests the producer to supply the data of the mapped application objects (see Table 3). The service is confirmed. The remote result parameter will confirm the value.

Table 3 - Read PDO

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
PDO Number	mandatory	
Remote Result		**Mandatory**
Data		mandatory

5.1.2.5 PDO protocol

5.1.2.5.1 Write PDO protocol

The service for a PDO write request is unconfirmed. The PDO producer sends the process data within a PDO to the network. There may be 0..n PDO consumers. At the PDO consumer(s) the reception of a valid PDO is indicated. Figure 10 shows the Write PDO Protocol.

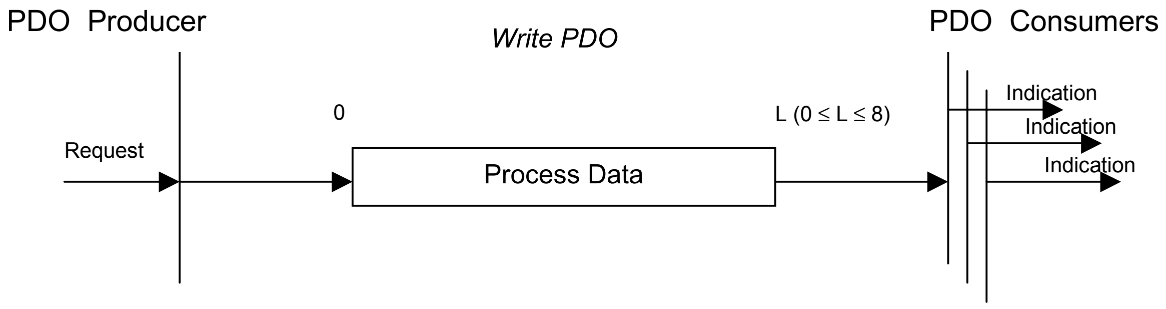


Figure 10 - Write PDO protocol

Process-Data: up to L bytes of application data according to the PDO mapping

If L exceeds the number of bytes ‘n’ defined by the actual PDO mapping length only the first ‘n’ bytes shall be used by the consumer. If L is less than ‘n’ the data of the received PDO shall not be processed and an Emergency message with error code 8210h shall be produced if Emergency is supported.

5.1.2.5.2 Read PDO protocol

The service for a PDO read request is confirmed. One or more PDO consumer transmit a remote transmission request frame (RTR) to the network. At the reception of the RTR frame the PDO producer of the requested PDO transmits the PDO. At all PDO consumers for this PDO the reception is indicated. There may be 0..n PDO consumers. The read service depends on the hardware capabilities. Figure 11 shows the Read PDO Protocol.

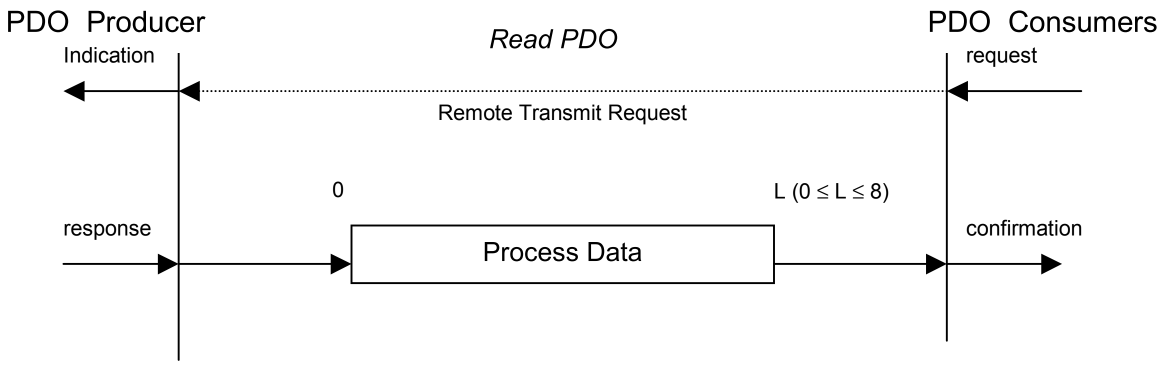


Figure 11 - Read PDO Protocol

Process-Data: up to L bytes of application data according to the PDO mapping

If L exceeds the number of bytes ‘n’ defined by the actual PDO mapping length only the first ‘n’ bytes shall be used by the consumer. If L is less than ‘n’ the data of the received PDO shall not be processed and an emergency message with error code 8210h shall be produced if emergency is supported.

5.1.3 Service Data Object (SDO)

5.1.3.1 General

With Service Data Objects (SDOs) the access to entries of a device object dictionary is provided. As these entries may contain data of arbitrary size and data type, SDOs may be used to transfer multiple *data sets* (each containing an arbitrary large block of data) from a client to a server and vice versa. The client may control via a *multiplexor* (index and sub-index of the object dictionary) which data set is to be transferred. The contents of the data set are defined within the Object dictionary.

Basically an SDO is transferred as a sequence of *segments*. Prior to transferring the segments there is an initialisation phase where client and server prepare themselves for transferring the segments. For SDOs, it is also possible to transfer a data set of up to four bytes during the initialisation phase. This mechanism is called an *expedited* transfer.

An SDO may be transferred as a sequence of *blocks* where each block is a sequence of up to 127 segments containing a sequence number and the data. Prior to transferring the blocks there is an initialisation phase where client and server may prepare themselves for transferring the blocks and negotiating the number of segments in one block. After transferring the blocks there is a finalisation phase where client and server may verify the correctness of the previous data transfer by comparing checksums derived from the data set. This transfer type mentioned above is called a *block* transfer which is faster than the segmented transfer for a large set of data.

In SDO Block Upload it is possible that the size of the data set does not justify the use of a block transfer because of the implied protocol overhead. In these cases a support for a fallback to the segmented or expedited transfer in initialisation phase may be implemented. As the assumption of the minimal data set size for which a block transfer outperforms the other transfer types depends on various parameters. The client indicates this threshold value in bytes to the server in the initialisation phase.

For the block transfer a Go-Back-n ARQ (Automatic Repeat Request) scheme is used to confirm each block.

After block download the server indicates the client the last successfully received segment of this block transfer by acknowledging this segment sequence number. Doing this the server implicitly acknowledges all segments preceding this segment. The client starts the following block transfer with the retransmission of all not acknowledged data. Additionally the server shall indicate the number of segments per block for the next block transfer.

After block upload the client indicates the server the last successfully received segment of this block transfer by acknowledging this segment sequence number. Doing this the client implicitly acknowledges all segments preceding this segment. The server shall start the following block transfer with the retransmission of all not acknowledged data. Additionally the client shall indicate the number of segments per block for the next block transfer.

For all transfer types it is the client that takes the initiative for a transfer. The owner of the accessed Object dictionary is the server of the SDO. Either client or server may take the initiative to abort the transfer of a SDO.

By means of an SDO a peer-to-peer communication channel between two devices is established. A device may support more than one SDO. One supported Server-SDO (SSDO) is the default case (Default SDO).

SDOs shall be described by the SDO communication parameter record (22h).The structure of this data type is given in annex A. The SDO communication parameter describes the communication capabilities of the Server-SDOs and Client-SDOs (CSDO). The indices of the corresponding Object dictionary entries shall be computed by the following formulas:

- SSDO communication parameter index = 1200h + SSDO-number –1;
- CSDO communication parameter index = 1280h + CSDO-number –1.

For each SDO the communication parameters shall be implemented. If only one SSDO exists the communication parameters may be omitted. The entries mentioned above are described in annex B.

5.1.3.2 SDO services

5.1.3.2.1 General

The model for the SDO communication shall be the client/server model as described in 4.4.3.

Attributes:

- SDO number:	SDO number [1..128] for every user type on the local device
- user type:	one of the values {client, server}
- mux data type:	multiplexor containing index and sub-index of type STRUCTURE OF UNSIGNED (16), UNSIGNED (8) , with index specifying an entry of the device object dictionary and "sub-index" specifying a component of a device object dictionary entry
- transfer type:	depends on the length of data to transfer: expedited for up to 4 data bytes segmented or block for more than 4 data bytes
- data type:	according to the referenced index and sub-index

The following services may be applied to an SDO depending on the application requirements:

- SDO Download, which may be split up into
  - Initiate SDO Download,
  - Download SDO Segment.
- SDO Upload, which may be split up into
  - Initiate SDO Upload,
  - Upload SDO Segment.
- Abort SDO Transfer

When using the segmented SDO download and upload services, the communication software will be responsible for transferring the SDO as a sequence of segments.

Expedited transfer shall be supported. Segmented transfer shall be supported if objects larger than 4 bytes are supported. The following SDO services for doing a block transfer with higher bus utilisation and performance for a large data set size may be implemented:

- SDO Block Download, which my be split up into
  - Initiate Block Download,
  - Download Block,
  - End Block Download.
- SDO Block Upload, which may be split up into
  - Initiate Block Upload,
  - Upload Block,
  - End Block Upload.

When using the SDO block download and upload services, the communication software will be responsible for transferring the data as a sequence of blocks.

In SDO Block Upload Protocol a support for a switch to SDO Upload Protocol in ‘Initiate SDO Block Upload’ may be implemented to increase transfer performance for data which size does not justifies using the protocol overhead of the ‘SDO Block Upload’ protocol.

For aborting an SDO block transfer the Abort SDO Transfer Service is used.

5.1.3.2.2 SDO download

Through this service (see Table 4) the client of an SDO downloads data to the server (owner of the Object dictionary). The data, the multiplexor (index and sub-index) of the data set that has been downloaded and its size (only for segmented transfer) are indicated to the server.

The service is confirmed. The remote result parameter will indicate the success or failure of the request. In case of a failure, the reason may be confirmed.

The SDO download shall consist of at least the Initiate SDO Download service and may consist of Download SDO Segment services (data length > 4 bytes).

Table 4 - SDO Download

Parameter	***Request / Indication***	***Response / Confirm***
Argument	**Mandatory**	
SDO Number	mandatory	
Data	mandatory	
Size	optional [1]	
Multiplexor	mandatory	
Remote Result		**Mandatory**
Success		selection
Failure		selection
Reason		optional

[1] *Optional* attributes may be implemented

5.1.3.2.3 Initiate SDO download

Through this service the client of SDO requests the server to prepare for downloading data to the server (see Table 5). The size of the data to be downloaded may be indicated to the server.

The multiplexor of the data set whose download is initiated and the transfer type are indicated to the server. In case of an expedited download, the data of the data set identified by the multiplexor and size is indicated to the server.

Table 5 - Initiate SDO Download

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO Number	mandatory	
Size	optional	
Multiplexor	mandatory	
Transfer type	mandatory	
Normal	selection	
Expedited	selection	
Data	mandatory	
Remote Result		**Mandatory**
Success		mandatory

The service is confirmed. The remote result parameter will indicate the success of the request. In case of a failure, an Abort SDO Transfer request shall be executed. In the case of a successful expedited download of a multiplexed DOMAIN, this service concludes the download of the data set identified by the multiplexor.

5.1.3.2.4 Download SDO segment

Through this service the client of a SDO supplies the data of the next segment to the server (see Table 6). The segment data shall be and its size may be indicated to the server. The continue parameter indicates the server whether there are still more segments to be downloaded or that this was the last segment to be downloaded.

Table 6 - Download SDO Segment

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO number	mandatory	
Data	mandatory	
Size	optional	
Continue	mandatory	
More	selection	
Last	selection	
Remote Result		**Mandatory**
Success		mandatory

The service is confirmed. The remote result parameter will indicate the success of the request. In case of a failure, an Abort SDO transfer request shall be executed. In case of success, the server has accepted the segment data and is ready to accept the next segment. There may be at most one Download SDO Segment service outstanding for an SDO transfer.

A successful Initiate SDO Download service with segmented transfer type shall have been executed prior to this service.

5.1.3.2.5 SDO upload

Through this service the client of a SDO uploads data from the server (see Table 7). The multiplexor (index and sub-index) of the data set that shall be uploaded is indicated to the server.

The SDO upload shall consist of at least the Initiate SDO Upload service and may consist of Upload SDO Segment services (data length > 4 bytes).

Table 7 - SDO Upload

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO number	mandatory	
Multiplexor	mandatory	
Remote Result		**Mandatory**
Success		selection
Data		mandatory
Size		optional
Failure		selection
Reason		optional

The service is confirmed. The remote result parameter will indicate the success or failure of the request. In case of a failure, the reason may be confirmed. In case of success, the data shall be confirmed and its size (for segmented transfer) may be confirmed.

5.1.3.2.6 Initiate SDO upload

Through this service the client of a SDO requests the server to prepare for uploading data to the client. The multiplexor (index and sub-index) of the data set whose upload is initiated is indicated to the server (see Table 8).

The service is confirmed. The remote result parameter will indicate the success of the request. In case of a failure, an Abort SDO Transfer request shall be executed. In the case of success, the size (for segmented transfer) of the data to be uploaded may be confirmed. In case of successful expedited upload, this service shall conclude the upload of the data set identified by multiplexor and the corresponding data shall be confirmed.

Table 8 - Initiate SDO Upload

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO number	mandatory	
Multiplexor	mandatory	
Remote Result		**Mandatory**
Success		mandatory
Size		optional
Multiplexor		mandatory
Transfer type		mandatory
Normal		selection
Expedited		selection
Data		mandatory

5.1.3.2.7 Upload SDO segment

Through this service the client of a SDO requests the server to supply the data of the next segment (see Table 9). The continue parameter indicates the client whether there are still more segments to be uploaded or that this was the last segment to be uploaded. There may be at most one Upload SDO Segment service outstanding for a SDO.

The service is confirmed. The remote result parameter will indicate the success of the request. In case of a failure, an Abort SDO Transfer request shall be executed. In case of success, the segment data shall be and its size may be confirmed.

A successful Initiate SDO Upload service with segmented transfer type shall have been executed prior to this service.

Table 9 - Upload SDO Segment

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO number	mandatory	
Remote Result		**Mandatory**
Success		mandatory
Data		mandatory
Size		optional
Continue		mandatory
More		selection
Last		selection

5.1.3.2.8 Abort SDO transfer

This service aborts the up- or download of a SDO referenced by its number (see Table 10). The reason may be indicated. The service is unconfirmed. Both the client and the server of a SDO may execute the service at any time. If the client of an SDO has a confirmed service outstanding, the indication of the abort is taken to be the confirmation of that service.

Table 10 - Abort SDO Transfer

Parameter	***Request / Indication***
Argument	**Mandatory**
SDO number	mandatory
Multiplexor	mandatory
Reason	mandatory

5.1.3.2.9 SDO block download

Through this service the client of SDO downloads data to the server of the SDO (owner of the Object dictionary) using the block download protocol (see Table 11). The data, the multiplexor (index and sub-index) of the data set that has been downloaded shall be indicated to the server and its size may be indicated as well.

The service is confirmed. The remote result parameter will indicate the success or failure of the request. In case of a failure, the reason may be confirmed.

Table 11 - SDO Block Download

Parameter	***Request / Indication***	***Response / Confirm***
Argument	**Mandatory**	
SDO Number	mandatory	
Data	mandatory	
Size	optional	
Multiplexor	mandatory	
Remote Result		**Mandatory**
Success		selection
Failure		selection
Reason		optional

5.1.3.2.10 Initiate SDO block download

Through this service the client of SDO requests the server of SDO (owner of the Object dictionary) to prepare for downloading data to the server (see Table 12).

The multiplexor of the data set whose download is initiated shall be and the size of the downloaded data in bytes may be indicated to the server.

The client as well as the server indicating their ability and/or demand to verify the complete transfer with a checksum in End SDO Block Download.

Table 12 - Initiate SDO Block Download

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO Number	mandatory	
Size	optional	
CRC ability	mandatory	
yes	selection	
no	selection	
Multiplexor	mandatory	
Remote Result		**Mandatory**
Success		mandatory
CRC ability		mandatory
yes		selection
no		selection
Blksize		mandatory

The service is confirmed. The remote result parameter will indicate the success of the request, the number of segments per block the server of SDO is able to receive and its ability and/or demand to verify the complete transfer with a checksum. In case of a failure, an Abort SDO Transfer request shall be executed.

5.1.3.2.11 Download SDO block

By this service the client of the SDO supplies the data of the next block to the server of the SDO (see Table 13). The block data is indicated to the server by a sequence of segments. Each segment consists of the data and a sequence number starting with 1 which is increased for each segment by 1 up to *blksize*. The parameter *blksize* is negotiated between server and client in the 'Initiate Block Download' protocol and may be changed by the server with each confirmation for a block transfer. The continue parameter indicates the server whether to stay in the 'Download Block' phase or to change in the 'End Download Block' phase.

Table 13 - Download SDO Block

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO Number	mandatory	
Data	mandatory	
Continue	mandatory	
More	selection	
Last	selection	
Remote Result		**Mandatory**
Success		mandatory
Ackseq		mandatory
Blksize		mandatory

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a success, the ackseq parameter indicates the sequence number of the last segment the server has received successfully. If this number does not correspond with the sequence number of the last segment sent by the client during this block transfer, the client shall retransmit all segments discarded by the server with the next block transfer. In case of a fatal failure, an Abort SDO Transfer request shall be executed. In case of success, the server has accepted all acknowledged segment data and is ready to accept the next block. There may be at most one Download SDO Block service outstanding for a SDO transfer.

A successful 'Initiate SDO Block Download' service shall have been executed prior to this service.

5.1.3.2.12 End SDO block download

Through this service the SDO Block Download is concluded (see Table 14).

The number of bytes not containing valid data in the last transmitted segments is indicated to the server.

If the server as well as the client have indicated their ability and demand to check the complete transfer with a checksum in ‘Initiate SDO Block Download’ this checksum is indicated to the server by the client. The server also shall generate a checksum which shall be compared with the one generated by the client.

Table 14 - End SDO Block Download

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO Number	mandatory	
Valid_data	mandatory	
Ckecksum	mandatory if negotiated in initiate	
Remote Result		**Mandatory**
Success		mandatory

The service is confirmed. The Remote Result parameter will indicate the success of the request (matching checksums between client and server if negotiated) and concludes the download of the data set. In case of a failure, an Abort SDO Transfer request shall be executed.

5.1.3.2.13 SDO block upload

Through this service the client of SDO uploads data from the server of SDO (owner of the Object dictionary) using the SDO block upload protocol (see Table 15). The data, the multiplexor (index and sub-index) of the data set that shall be uploaded and its size may be indicated to the server.

The service is confirmed. The Remote Result parameter shall indicate the success or failure of the request. In case of a failure, the reason may be confirmed. In case of a success, the data shall be confirmed and its size may be confirmed.

Table 15 - SDO Block Upload

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO Number	mandatory	
Multiplexor	mandatory	
Remote Result		**Mandatory**
Success		selection
Data		mandatory
Size		optional
Failure		selection
Reason		optional

5.1.3.2.14 Initiate SDO block upload

Through this service the client of SDO requests the server of SDO (owner of the Object dictionary) to prepare for uploading data to the client (see Table 16).

The multiplexor of the data set whose upload is initiated and the number of segments the client of the SDO is able to receive are indicated to the server.

A protocol switch threshold value is indicated to the server. If the number of bytes that shall be uploaded is less or equal this value the server may conclude this data transfer with the ‘SDO Upload Protocol’ as described in 5.1.3.2.5.

The client as well as the server indicating their ability and/or demand to verify the complete transfer with a checksum in End SDO Block Upload

The size of the uploaded data in bytes may be indicated to the client.

Table 16 - Initiate SDO Block Upload

Parameter	***Request / Indication***	***Response / Confirm***
Argument	**Mandatory**	
SDO Number	mandatory	
Blksize	mandatory	
CRC ability	mandatory	
Yes	selection	
No	selection	
Multiplexor	mandatory	
Threshold	mandatory	
Remote Result		**Mandatory**
Success		mandatory
CRC ability		mandatory
Yes		selection
No		selection
Size		optional

The service is confirmed. In case of a failure, an Abort SDO Transfer request shall be executed. In case of success the size of the data in bytes to be uploaded may be indicated to the client.

If the size of the data that shall be uploaded is less or equal threshold the server may continue with the SDO block upload protocol. In this case the Remote Result parameter and the further protocol is described in 5.1.3.3.15.

5.1.3.2.15 Upload SDO block

Through this service the client of SDO uploads the data of the next block from the server of the SDO (see Table 17). The block data is indicated to the client by a sequence of segments. Each segment consists of the segment data and a sequence number starting with 1 which is increased for each segment by 1 up to blksize. The parameter blksize is negotiated between server and client in the 'Initiate Block Upload' protocol and may be changed by the client with each confirmation for a block transfer. The continue parameter indicates the client whether to stay in the 'Upload Block' phase or to change in the 'End Upload Block' phase.

Table 17 - Upload SDO Block

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO Number	mandatory	
Data	mandatory	
Continue	mandatory	
More	selection	
Last	selection	
Remote Result		**Mandatory**
Success		mandatory
Ackseq		mandatory
Blksize		mandatory

The service is confirmed. The Remote Result parameter will indicate the success of the request. In case of a success the ackseq parameter indicates the sequence number of the last segment the client has received successfully. If this number does not correspond with the sequence number of the last segment sent by the server during this block transfer the server shall retransmit all segments discarded by the client with the next block transfer. In case of a fatal failure, an Abort SDO Transfer request shall be executed. In case of success, the client has accepted all acknowledged segment data and is ready to accept the next block. There may be at most one Upload Block service outstanding for a SDO transfer.

A successful 'Initiate SDO Block Upload' service shall have been executed prior to this service.

5.1.3.2.16 End SDO block upload

Through this service the SDO Block Upload is concluded (see Table 18).

The number of bytes not containing valid data in the last transmitted segments is indicated to the client.

If the server as well as the client have indicated their ability and demand to check the complete transfer with a checksum during ‘Initiate SDO Block Upload’ this checksum is indicated to the client by the server. The client also shall generate a checksum which shall be compared with the one generated by the server.

Table 18 - End SDO Block Upload

Parameter	*Request / Indication*	*Response / Confirm*
Argument	**Mandatory**	
SDO Number	mandatory	
Valid_data	mandatory	
Checksum	mandatory if negotiated in initiate	
Remote Result		**Mandatory**
Success		mandatory

The service is confirmed. The Remote Result parameter will indicate the success of the request (matching checksums between client and server if demanded) and concludes the download of the data set. In case of a failure, an Abort SDO Transfer request shall be executed.

5.1.3.3 SDO protocols

5.1.3.3.1 General

Six confirmed services (SDO Download, SDO Upload, Initiate SDO Upload, Initiate SDO Download, Download SDO Segment, and Upload SDO Segment) and one unconfirmed service (Abort SDO Transfer) are defined for Service Data Objects doing the segmented/expedited transfer.

Eight confirmed services (SDO Block Download, SDO Block Upload, Initiate SDO Upload, Initiate SDO Block Download, Download SDO Segment, Upload SDO Segment, End SDO Upload and End SDO Block Download) and one unconfirmed service (Abort SDO Block Transfer) are defined for Service Data Objects doing the optional block transfer.

5.1.3.3.2 Download SDO protocol

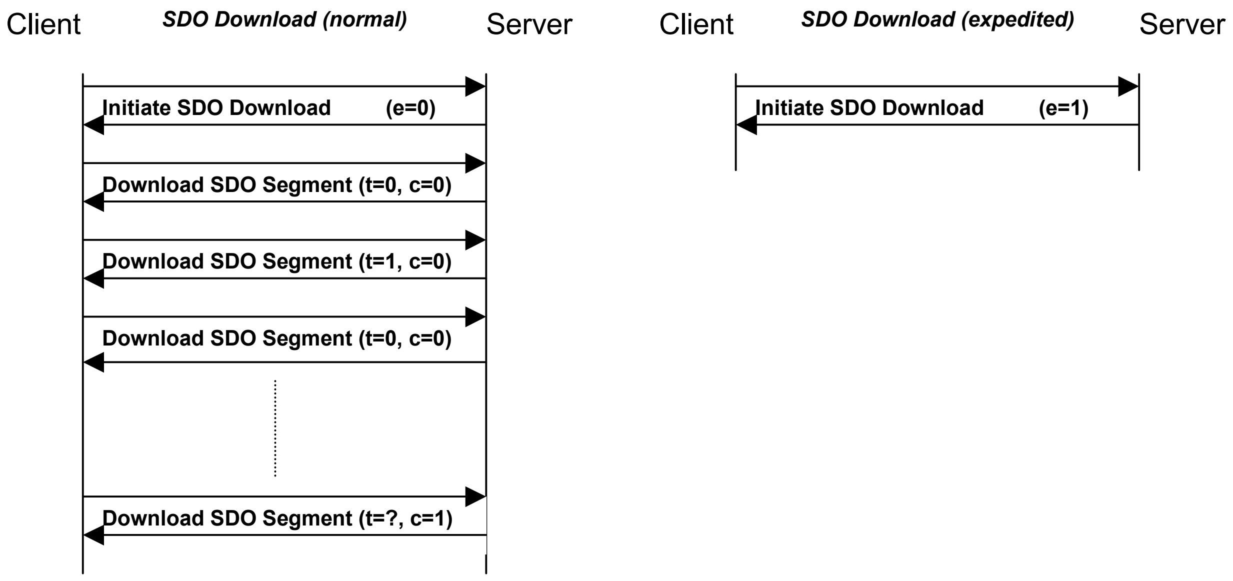

Client
SDO Download (normal)
Server
Client
SDO Download (expedited)
Server
Initiate SDO Download (e=0)
Download SDO Segment (t=0, c=0)
Download SDO Segment (t=1, c=0)
Download SDO Segment (t=0, c=0)
Download SDO Segment (t=?, c=1)
Initiate SDO Download (e=1)

Figure 12 - Download SDO Protocol

This protocol shall be used to implement the SDO Download service (see Figure 12). SDOs are downloaded as a sequence of zero or more Download SDO Segment services preceded by an Initiate SDO Download service. The sequence shall be terminated by

- an Initiate SDO Download request/indication with the e-bit set to 1 followed by an Initiate SDO Download response/confirm, indicating the successful completion of an expedited download sequence,
- a Download SDO Segment response/confirm with the c-bit set to 1, indicating the successful completion of a normal download sequence,
- an Abort SDO Transfer request/indication, indicating the unsuccessful completion of the download sequence,
- a new Initiate Domain Download request/indication, indicating the unsuccessful completion of the download sequence and the start of a new download sequence.

If in the download of two consecutive segments the toggle bit does not alter, the contents of the last segment shall be ignored. If such an error is reported to the application, the application may decide to abort the download.

5.1.3.3.3 Initiate SDO download protocol

This protocol shall be used to implement the Initiate SDO Download service for SDOs (see Figure 13).

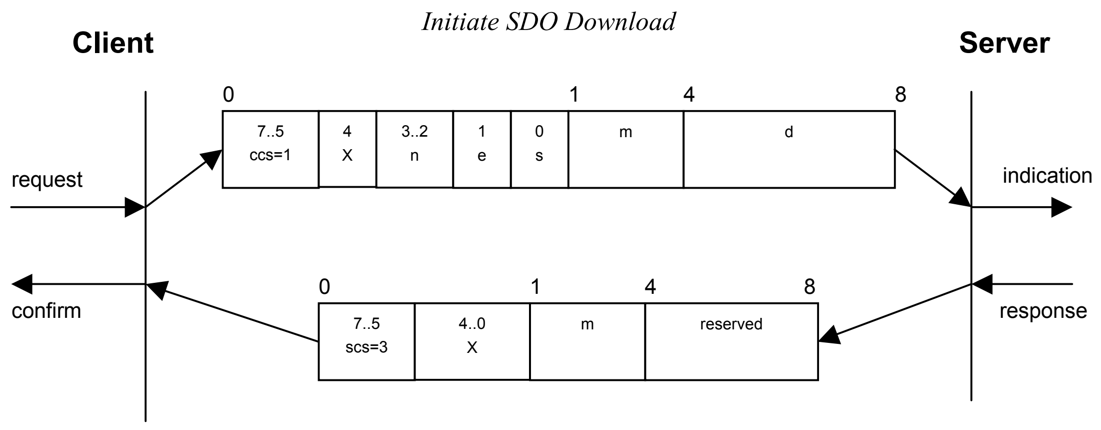


Figure 13 - Initiate SDO Download Protocol

- **ccs:** client command specifier

  1: initiate download request
- **scs:** server command specifier

  3: initiate download response
- **n:** Only valid if e = 1 and s = 1, otherwise 0. If valid it indicates the number of bytes in d that do not contain data. Bytes [8-n, 7] do not contain data.
- **e**: transfer type

  0: normal transfer

  1: expedited transfer
- **s:** size indicator

  0: data set size is not indicated

  1: data set size is indicated
- **m**: multiplexor. It represents the index/sub-index of the data to be transferred by the SDO.
- **d:** data

  e = 0, s = 0:d is reserved for further use.

  e = 0, s = 1:d contains the number of bytes to be downloaded.

  Byte 4 contains the lsb and byte 7 contains the msb.

  e = 1, s = 1: d contains the data of length 4-n to be downloaded,

  the encoding depends on the type of the data referenced

  by index and sub-index

  e = 1, s = 0:d contains unspecified number of bytes to be downloaded
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0

5.1.3.3.4 Download SDO segment protocol

This protocol shall be used to implement the Download SDO Segment service (see Figure 14).

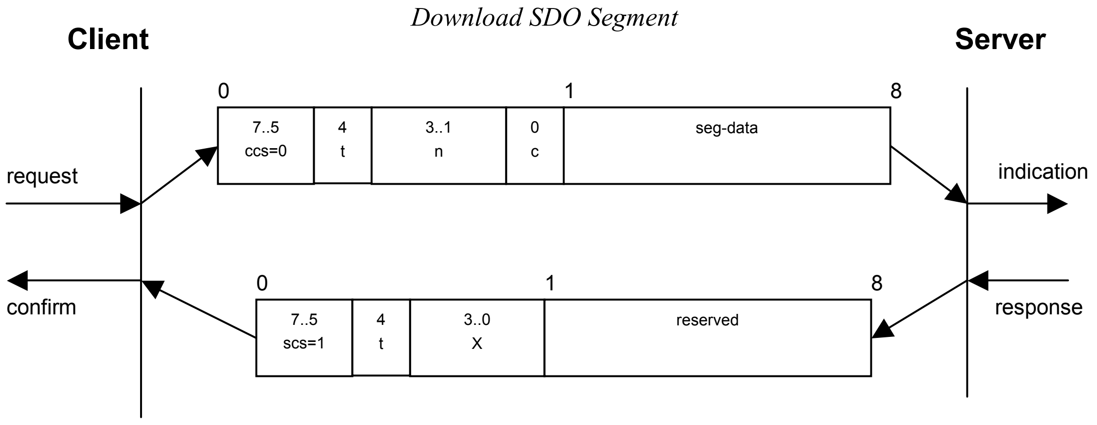


Figure 14 - Download SDO Segment Protocol

- **ccs:** client command specifier

  0: download segment request
- **scs:** server command specifier

  1: download segment response
- **seg-data:** at most 7 bytes of segment data to be downloaded. The encoding depends on the type of the data referenced by index and sub-index
- **n:** indicates the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.
- **c:** indicates whether there are still more segments to be downloaded.

  0 more segments to be downloaded

  1: no more segments to be downloaded
- **t:** toggle bit. This bit shall alternate for each subsequent segment that is downloaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0

5.1.3.3.5 Upload SDO protocol

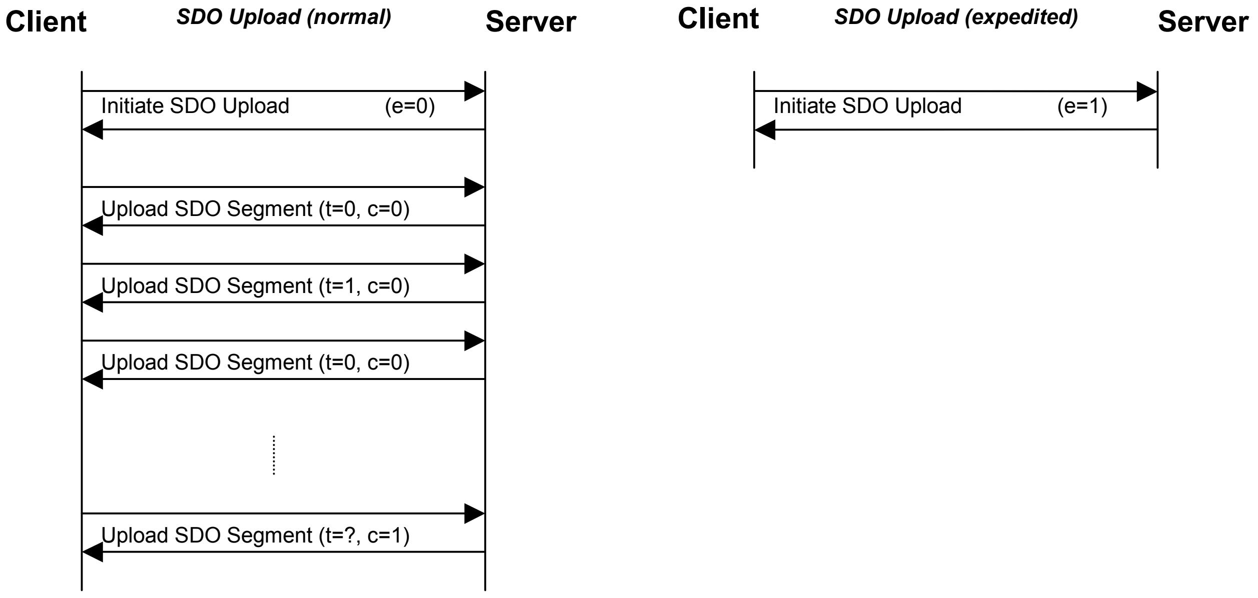

Client
SDO Upload (normal)
Server
Initiate SDO Upload (e=0)
Upload SDO Segment (t=0, c=0)
Upload SDO Segment (t=1, c=0)
Upload SDO Segment (t=0, c=0)
Upload SDO Segment (t=?, c=1)
Client
SDO Upload (expedited)
Server
Initiate SDO Upload (e=1)

Figure 15 - Upload SDO Protocol

This protocol shall be used to implement the SDO Upload service (see Figure 15). SDO are uploaded as a sequence of zero or more Upload SDO Segment services preceded by an Initiate SDO Upload service. The sequence shall be terminated by

- an Initiate SDO Upload response/confirm with the e-bit set to 1, indicating the successful completion of an expedited upload sequence,
- an Upload SDO Segment response/confirm with the c-bit set to 1, indicating the successful completion of a normal upload sequence,
- an Abort SDO Transfer request/indication, indicating the unsuccessful completion of the upload sequence,
- a new Initiate SDO Upload request/indication, indicating the unsuccessful completion of the upload sequence and the start of a new sequence.

If in the upload of two consecutive segments the toggle bit does not alter, the contents of the last segment shall be ignored. If such an error is reported to the application, the application may decide to abort the upload.

5.1.3.3.6 5.1.3.3.6 Initiate SDO upload protocol

This protocol shall be used to implement the Initiate SDO Upload service (see Figure 16).

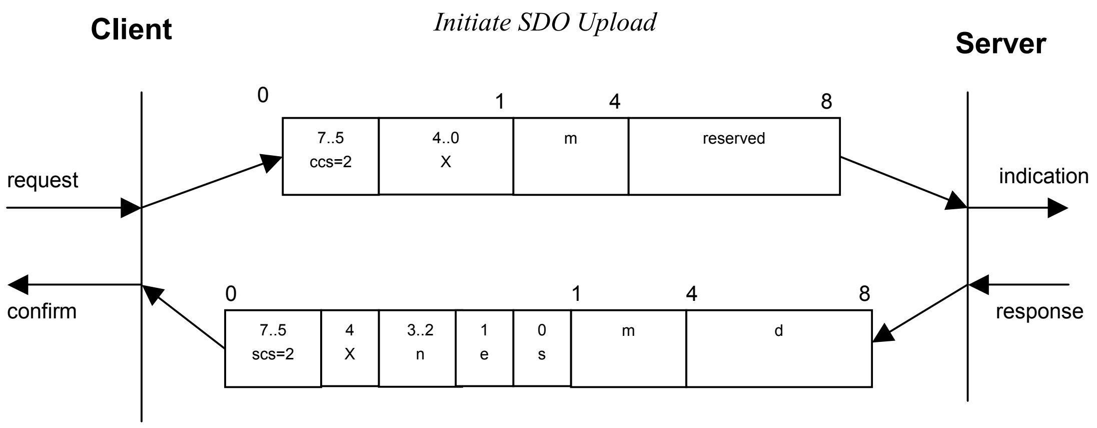


Figure 16 - Initiate SDO Upload Protocol

- **ccs:** client command specifier

  2: initiate upload request
- **scs:** server command specifier

  2: initiate upload response
- **n:** Only valid if e = 1 and s = 1, otherwise 0. If valid it indicates the number of bytes in d that do not contain data. Bytes [8-n, 7] do not contain segment data.
- **e**: transfer type

  0: normal transfer

  1: expedited transfer
- **s:** size indicator

  0: data set size is not indicated

  1: data set size is indicated
- **m**: multiplexor. It represents the index/sub-index of the data to be transferred by the SDO.
- **d:** data

  e = 0, s = 0:d is reserved for further use.

  e = 0, s = 1:d contains the number of bytes to be uploaded.

  Byte 4 contains the lsb and byte 7 contains the msb.

  e = 1, s = 1:d contains the data of length 4-n to be uploaded,

  the encoding depends on the type of the data referenced

  by index and sub-index

  e = 1, s = 0:d contains unspecified number of bytes to be uploaded.
- **X:** not used, always 0
- **reserved:** reserved for further use , always 0

5.1.3.3.7 Upload SDO segment protocol

This protocol shall be used to implement the Upload SDO Segment service (see Figure 17).

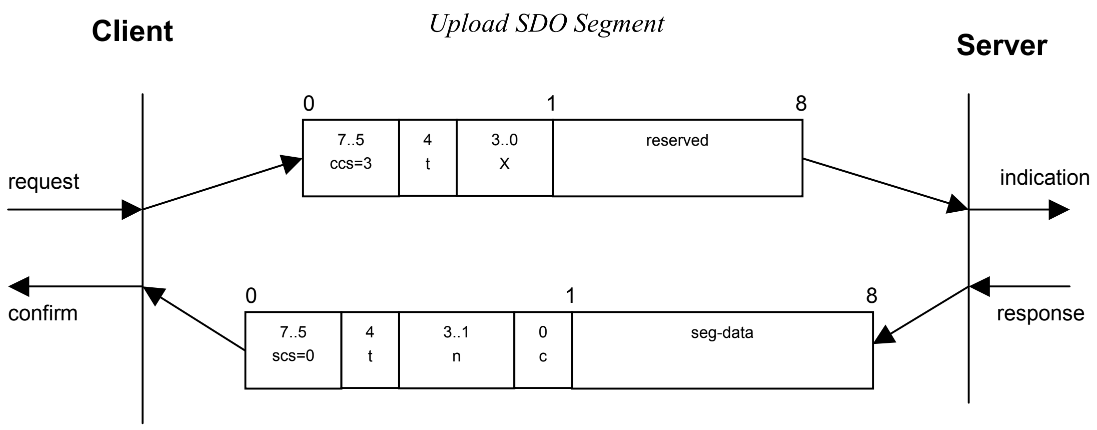


Figure 17 - Upload SDO Segment Protocol

- **ccs:** client command specifier

  3: upload segment request
- **scs:** server command specifier

  0: upload segment response
- **t:** toggle bit. This bit shall alternate for each subsequent segment that is uploaded. The first segment will have the toggle-bit set to 0. The toggle bit will be equal for the request and the response message.
- **c:** indicates whether there are still more segments to be uploaded.

  0: more segments to be uploaded

  1: no more segments to be uploaded
- **seg-data:** at most 7 bytes of segment data to be uploaded. The encoding depends on the type of the data referenced by index and sub-index
- **n:** indicates the number of bytes in seg-data that do not contain segment data. Bytes [8-n, 7] do not contain segment data. n = 0 if no segment size is indicated.
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0

5.1.3.3.8 Abort SDO transfer protocol

This protocol shall be used to implement the Abort SDO Transfer Service (see Figure 18).

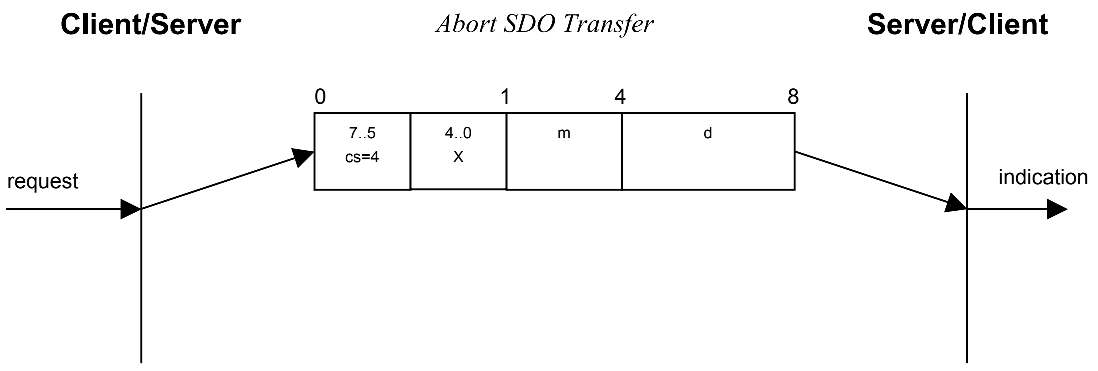


Figure 18 - Abort SDO Transfer Protocol

- **cs:** command specifier
  4: abort transfer request
- **X:** not used, always 0
- **m**: multiplexor. It represents index and sub-index of the SDO.
- **d:** contains a 4 byte abort code about the reason for the abort.

The abort code shall be encoded as UNSIGNED32 value accordingly to Table 19.

Table 19 - SDO abort codes

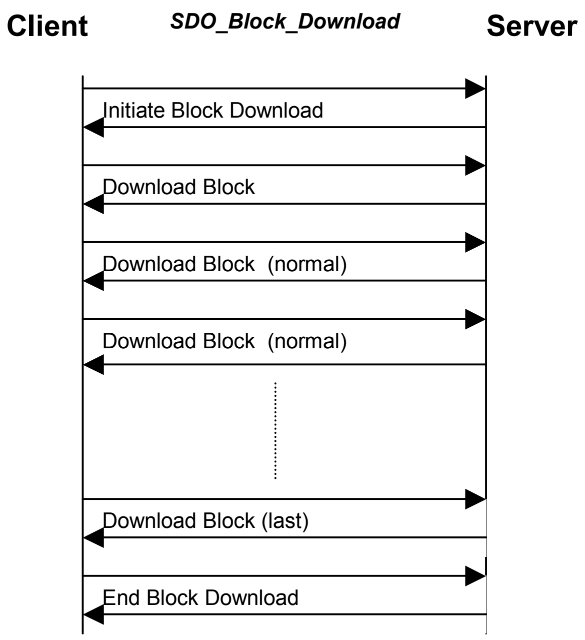
Abort code	**Description**
0503 0000h	Toggle bit not alternated.
0504 0000h	SDO protocol timed out.
0504 0001h	Client/server command specifier not valid or unknown.
0504 0002h	Invalid block size (block mode only).
0504 0003h	Invalid sequence number (block mode only).
0504 0004h	CRC error (block mode only).
0504 0005h	Out of memory.
0601 0000h	Unsupported access to an object.
0601 0001h	Attempt to read a write only object.
0601 0002h	Attempt to write a read only object.
0602 0000h	Object does not exist in the object dictionary.
0604 0041h	Object cannot be mapped to the PDO.

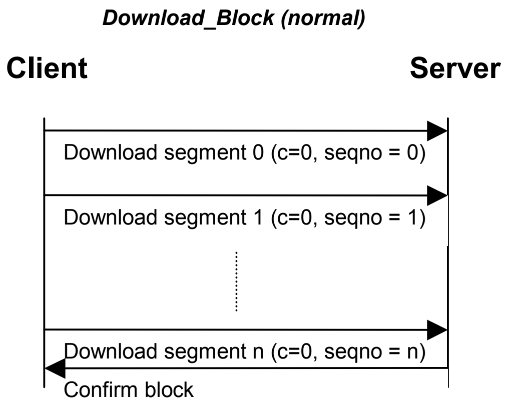
Table 19 - SDO abort codes (continued)

Abort code	**Description**
0604 0042h	The number and length of the objects to be mapped would exceed PDO length.
0604 0043h	General parameter incompatibility reason.
0604 0047h	General internal incompatibility in the device.
0606 0000h	Access failed due to an hardware error.
0607 0010h	Data type does not match, length of service parameter does not match
0607 0012h	Data type does not match, length of service parameter too high
0607 0013h	Data type does not match, length of service parameter too low
0609 0011h	Sub-index does not exist.
0609 0030h	Value range of parameter exceeded (only for write access).
0609 0031h	Value of parameter written too high.
0609 0032h	Value of parameter written too low.
0609 0036h	Maximum value is less than minimum value.
0800 0000h	general error
0800 0020h	Data cannot be transferred or stored to the application.
0800 0021h	Data cannot be transferred or stored to the application because of local control.
0800 0022h	Data cannot be transferred or stored to the application because of the present device state.
0800 0023h	Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error).

The abort codes not listed are reserved.

5.1.3.3.9 SDO Block download protocol




Download_Block (normal)
Client
Server
Download segment 0 (c=0, seqno = 0)
Download segment 1 (c=0, seqno = 1)
Download segment n (c=0, seqno = n)
Confirm block

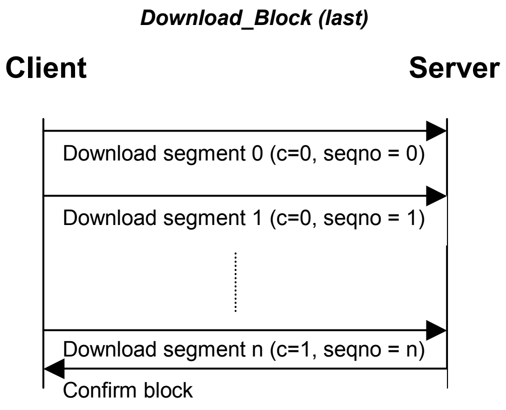

Download_Block (last)
Client
Server
Download segment 0 (c=0, seqno = 0)
Download segment 1 (c=0, seqno = 1)
Download segment n (c=1, seqno = n)
Confirm block

Figure 19 - SDO Block Download Protocol

This protocol shall be used to implement a SDO Block Download service (see Figure 19). SDOs are downloaded as a sequence of Download SDO Block services preceded by an Initiate SDO Block Download service. The SDO Download Block sequence is terminated by

- a downloaded segment within a block with the c-bit set to 1, indicating the completion of the block download sequence,
- an 'Abort SDO Transfer' request/indication, indicating the unsuccessful completion of the download sequence.

The whole ‘SDO Block Download’ service shall be terminated with the End SDO Block Download service. If client as well as server have indicated the ability to generate a CRC during the Initiate SDO Block Download service the server shall generate the CRC on the received data. If this CRC differs from the CRC generated by the client the server shall indicate this with an ‘Abort SDO Transfer’ indication.

5.1.3.3.10 Initiate SDO block download protocol

This protocol shall be used to implement the Initiate SDO Block Download service (see Figure 20).

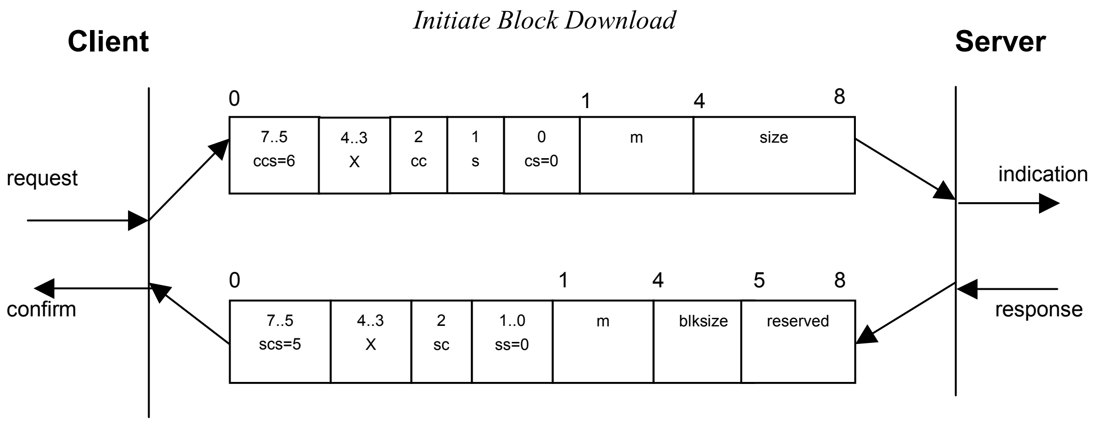


Figure 20 - Initiate SDO Block Download Protocol

- **ccs:** client command specifier

  6: block download

- **scs:** server command specifier

  5: block download

- **s:** size indicator

  0: data set size is not indicated

  1: data set size is indicated

- **cs:** client subcommand

  0: initiate download request

- **ss:** server subcommand

  0: initiate download response

- **cc**: client CRC support

  cc = 0: Client does not support generating CRC on data

  cc = 1: Client supports generating CRC on data

- **sc**: server CRC support

  sc = 0: Server does not support generating CRC on data

  sc = 1: Server supports generating CRC on data

- **m**: multiplexor. It represents the index/sub-index of the data to be transfer by the SDO.
- **size:** download size in bytes

  s = 0: size is reserved for further use, always 0

  s = 1: size contains the number of bytes to be downloaded

  Byte 4 contains the lsb and byte 7 the msb

- **blksize:** Number of segments per block with 0 < blksize < 128.
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0

5.1.3.3.11 Download SDO block segment protocol

This protocol shall be used to implement the Block Download service (see Figure 21).

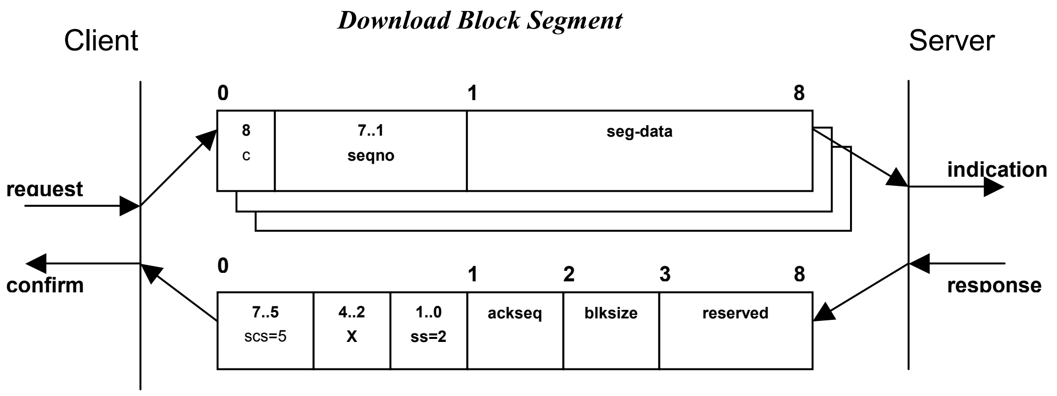


Figure 21 - Download SDO Block Segment

- **scs:** server command specifier

  5: block download
- **ss**: server subcommand

  2: block download response
- **c:** indicates whether there are still more segments to be downloaded

  0: more segments to be downloaded

  1: no more segments to be downloaded, enter End SDO block download phase
- **seqno**: sequence number of segment 0 < seqno < 128.
- **seg-data**: at most 7 bytes of segment data to be downloaded.
- **ackseq**: sequence number of last segment that was received successfully during the last block download. If ackseq is set to 0 the server indicates the client that the segment with the sequence number 1 was not received correctly and all segments shall be retransmitted by the client.
- **blksize:** Number of segments per block that shall be used by client for the following block download with 0 < blksize < 128.
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0.

5.1.3.3.12 End SDO block download protocol

This protocol shall be used to implement the End SDO Block Download service (see Table 22).

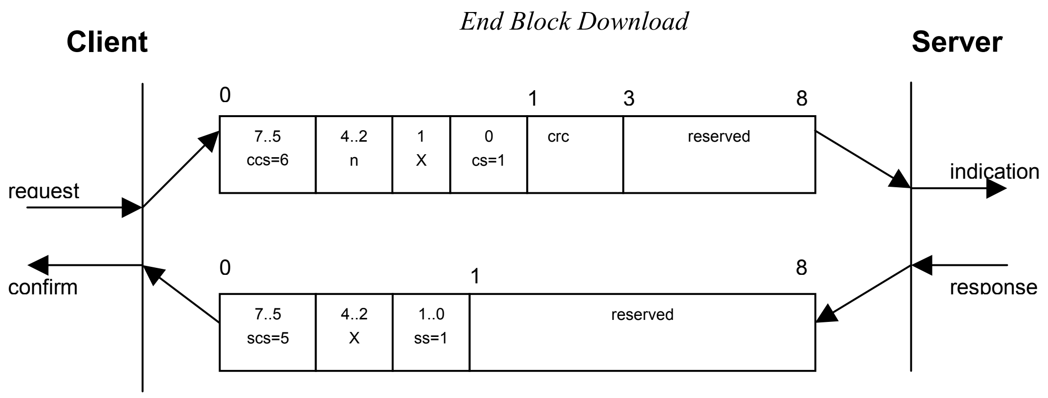


Figure 22 - End SDO Block Download Protocol

- **ccs:** client command specifier

  6: block download
- **scs:** server command specifier

  5: block download
- **cs:** client subcommand

  1: end block download request
- **ss:** server subcommand

  1: end block download response
- **n**: indicates the number of bytes in the last segment of the last block that do not contain data. Bytes [8-n, 7] do not contain segment data.
- **crc**: 16 bit Cyclic Redundancy Checksum (CRC) for the whole data set. The algorithm for generating the CRC is described in 5.1.3.3.17. CRC is only valid if in Initiate Block Download cc and sc are set to 1 otherwise CRC shall be set to 0.
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0

5.1.3.3.13 Upload SDO block protocol

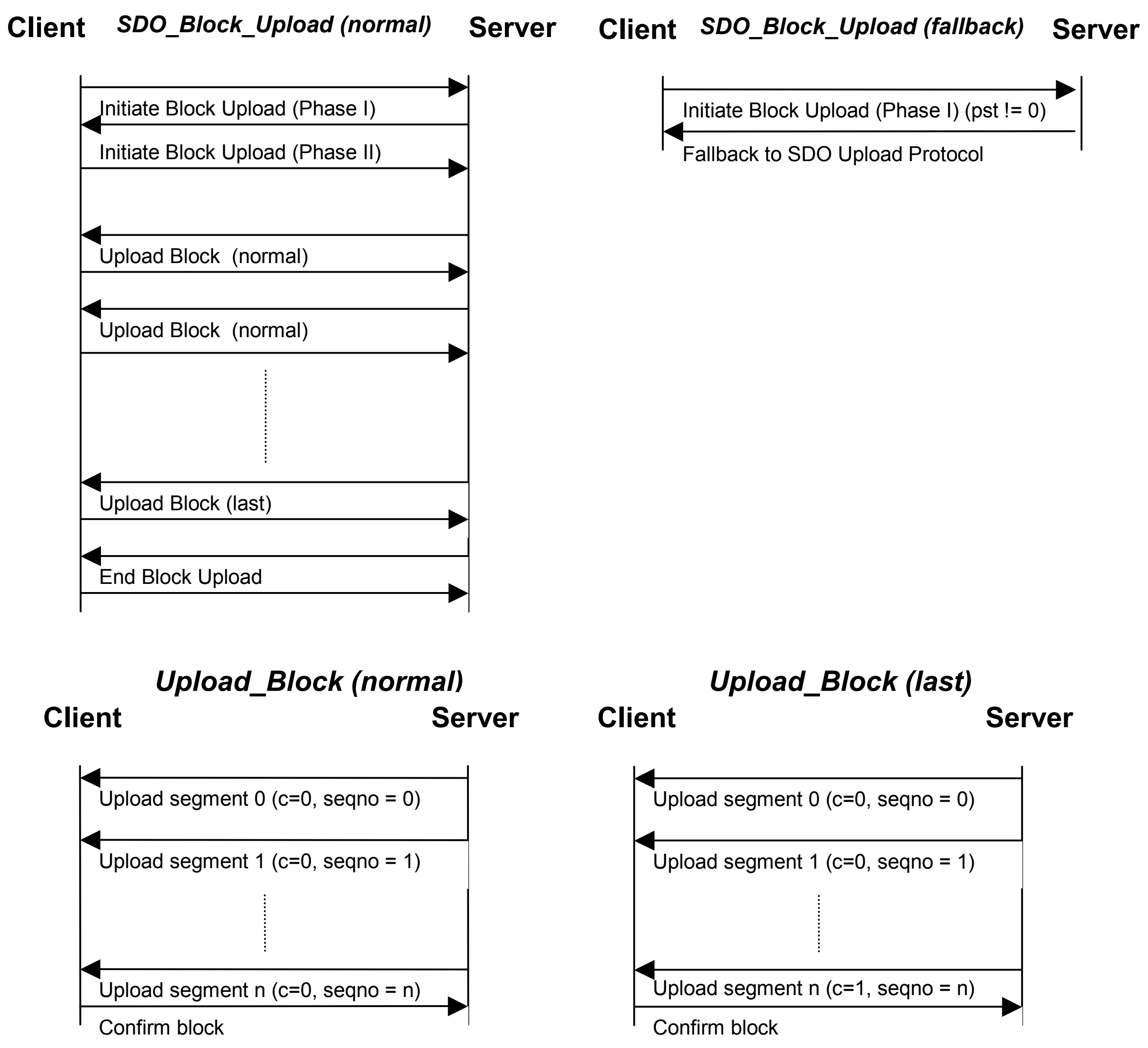

Client
SDO_Block_Upload (normal)
Server
Initiate Block Upload (Phase I)
Initiate Block Upload (Phase II)
Upload Block (normal)
Upload Block (normal)
Upload Block (last)
End Block Upload
Client
SDO_Block_Upload (fallback)
Server
Initiate Block Upload (Phase I) (pst != 0)
Fallback to SDO Upload Protocol
Upload_Block (normal)
Client
Server
Upload segment 0 (c=0, seqno = 0)
Upload segment 1 (c=0, seqno = 1)
Upload segment n (c=0, seqno = n)
Confirm block
Upload_Block (last)
Client
Server
Upload segment 0 (c=0, seqno = 0)
Upload segment 1 (c=0, seqno = 1)
Upload segment n (c=1, seqno = n)
Confirm block

Figure 23 - Upload SDO Block Protocol

This protocol shall be used to implement a SDO Block Upload service which starts with the Initiate SDO Block Upload service (see Figure 23). The client may indicate a threshold value to the server which is the minimum value in bytes to increase transfer performance using the SDO Block Upload protocol instead of the SDO Upload protocol. If the data set size is less or equal this value the server may continue with the normal or expedited transfer of the SDO Block Upload protocol.

Otherwise SDOs are uploaded as a sequence of Upload SDO Block services. The SDO Upload Block sequence is terminated by

- an uploaded segment within a block with the c-bit set to 1, indicating the completion of the block upload sequence,
- an Abort SDO Transfer request/indication, indicating the unsuccessful completion of the uploaded sequence.

The whole ‘SDO Block Upload’ service is terminated with the ‘End SDO Block Upload’ service. If client as well as server have indicated the ability to generate a CRC during the Initiate SDO Block Upload service the client shall generate the CRC on the received data. If this CRC differs from the CRC generated by the server the client shall indicate this with an ‘Abort SDO Transfer’ indication.

5.1.3.3.14 Initiate SDO Block upload protocol

This protocol shall be used to implement the Initiate SDO Block Upload service (see Figure 24). If the value of the Protocol Switch Threshold parameter indicated by the client in the first request is less or equal the data set size to be uploaded the server may continue with the SDO Upload Protocol as described in 5.1.3.3.5.

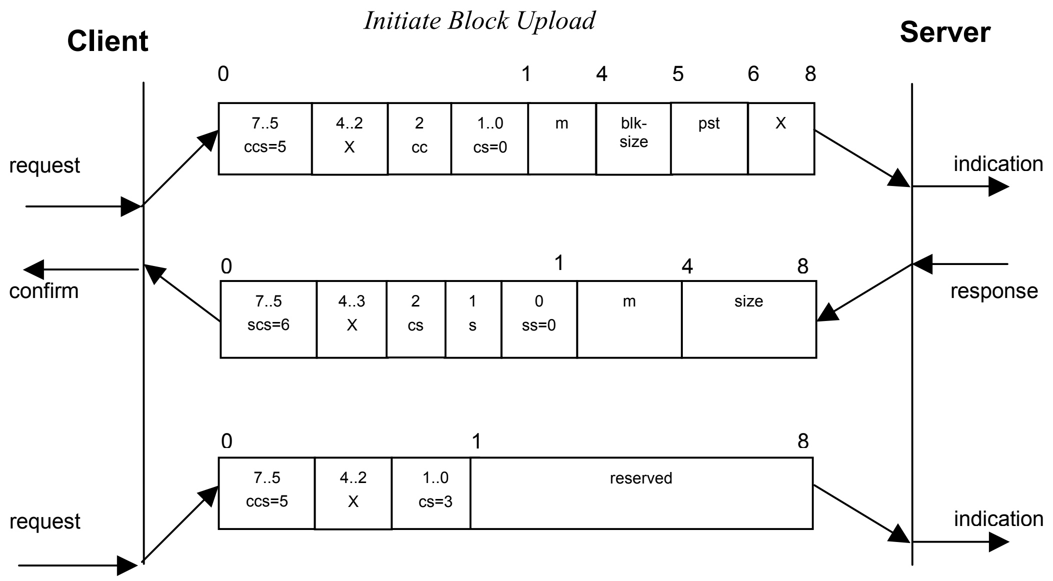


Figure 24 - Initiate SDO Block Upload Protocol

- **ccs**: client command specifier
  5: block upload
- **scs:** server command specifier
  6: block upload
- **cs**: client subcommand
  0: initiate upload request
  3: start upload
- **ss**: server subcommand
  0: initiate upload response
- **m**: multiplexor. It represents the index/sub-index of the data to be transfer by the SDO.
- **cc**: client CRC support
  cc = 0: Client does not support generating CRC on data
  cc = 1: Client supports generating CRC on data
- **sc**: server CRC support
  sc = 0: Server does not support generating CRC on data
  sc = 1: Server supports generating CRC on data
- **pst**: Protocol Switch Threshold in bytes to change the SDO transfer protocol
  pst = 0: Change of transfer protocol not allowed.
  pst > 0: If the size of the data in bytes that shall be uploaded is less or equal pst the server may switch to the ‘SDO Upload Protocol’ by transmitting the server response of the ‘SDO Upload Protocol’ as described in 5.1.3.3.5.

- **s**: size indicator
  0: data set size is not indicated
  1: data set size is indicated
- **size:** upload size in bytes

s = 0:	size is reserved for further use, always 0
s = 1:	size contains the number of bytes to be downloaded Byte 4 contains the lsb and byte 7 the msb

- **blksize:** Number of segments per block with 0 < blksize < 128.
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0

5.1.3.3.15 Upload SDO block segment protocol

This protocol shall be used to implement the SDO Block Upload service (see Figure 25).

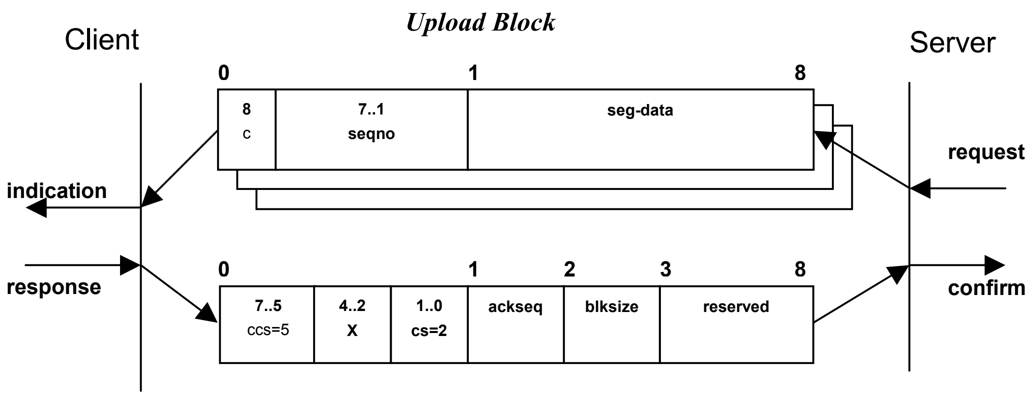


Figure 25 - Upload SDO Block Segment Protocol

- **ccs:** server command specifier
  5: block upload
- **cs:** client subcommand
  2: block upload response
- **c:** indicates whether there are still more segments to be downloaded
  0: more segments to be uploaded
  1: no more segments to be uploaded, enter ‘End block upload’ phase
- **seqno**: sequence number of segment 0 < seqno < 128.
- **seg-data:** at most 7 bytes of segment data to be uploaded.
- **ackseq:** sequence number of last segment that was received successfully during the last block upload. If ackseq is set to 0 the client indicates the server that the segment with the sequence number 1 was not received correctly and all segments shall be retransmitted by the server.
- **blksize:** Number of segments per block that shall be used by server for the following block upload with 0 < blksize < 128.
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0

5.1.3.3.16 End SDO block upload protocol

This protocol shall be used to implement the End SDO Block Upload service (see Figure 26).

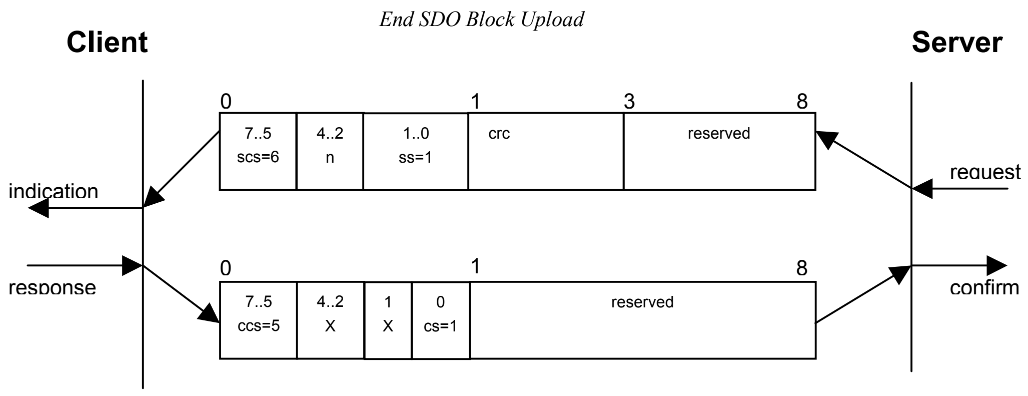


Figure 26 - End SDO Block Upload Protocol

- **ccs:** client command specifier

  5: block upload
- **scs:** server command specifier

  6: block upload
- **cs:** client subcommand

  1: end block upload request
- **ss:** server subcommand

  1: end block upload response
- **n**: indicates the number of bytes in the last segment of the last block that do not contain data. Bytes [8-n, 7] do not contain segment data.
- **crc:** 16 bit Cyclic Redundancy Checksum (CRC) for the whole data set. The algorithm for generating the CRC is described in 5.1.3.3.17. CRC is only valid if in Initiate Block Upload cc and sc are set to 1 otherwise crc shall be set to 0.
- **X:** not used, always 0
- **reserved:** reserved for further use, always 0

5.1.3.3.17 CRC calculation algorithm to verify SDO block transfer

To verify the correctness of a SDO block upload/download client and server may calculate a cyclic redundancy checksum (CRC) which is exchanged and verified during End SDO Block Download/Upload protocol. The check polynominal shall be x^16 + x^12 + x^5 + 1.

5.1.4 Synchronisation object (SYNC)

5.1.4.1 General

The Synchronisation Object is broad-cast periodically by the SYNC producer. This SYNC provides the basic network clock. The time period between two consecutive SYNCs is specified by the standard parameter communication cycle period (see Object 1006h: Communication Cycle Period), which may be written by a configuration tool to the application devices during the boot-up process. There may be a time jitter in transmission by the SYNC producer corresponding approximately to the latency due to some other message being transmitted just before the SYNC.

In order to guarantee timely access to the CAN bus the SYNC is given a very high priority identifier (1005h). Devices which operate synchronously may use the SYNC object to synchronise their own timing with that of the Synchronisation Object producer. The details of this synchronisation are device dependent and do not fall within the scope of this European Standard. Devices which require a more accurate common time base may use the high resolution synchronisation mechanism described in 5.2.2.

5.1.4.2 SYNC services

The SYNC transmission follows the producer/consumer push model as described in 4.4.4. The service is unconfirmed.

Attributes:

- user type:	one of the values {consumer, producer}
- data type:	nil

5.1.4.3 SYNC protocol

One unconfirmed service (Write SYNC) is defined (see Figure 27).

Write SYNC

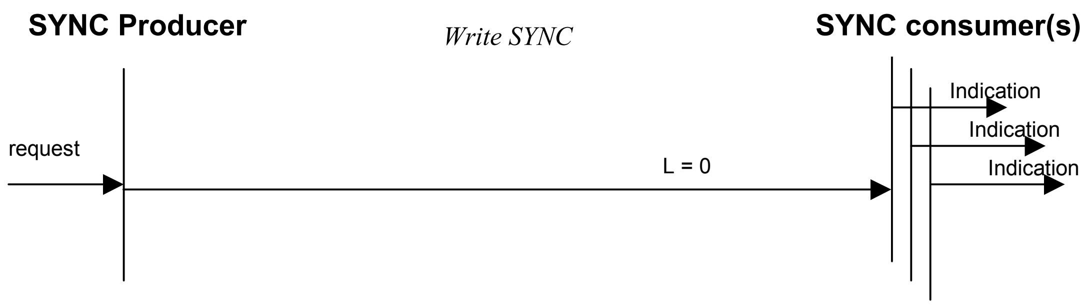


Figure 27 - SYNC Protocol

- The SYNC does not carry any data (L=0).

The Identifier of the SYNC object shall be located at Object Index 1005h.

5.1.5 Time stamp object (TIME)

5.1.5.1 General

By means of the Time Stamp Object a common time frame reference is provided to devices. It shall contain a value of the type TIME_OF_DAY. The Identifier of the TIME Object shall be located at Object Index 1012h.

5.1.5.2 TIME services

The Time Stamp Object transmission follows the producer/consumer push model as described in 4.4.4. The service is unconfirmed.

Attributes:

- user type:	one of the values {consumer, producer}
- data type:	TIME_OF_DAY

5.1.5.3 TIME protocol

One unconfirmed service (Write TIME) is defined (see Figure 28).

Write TIME

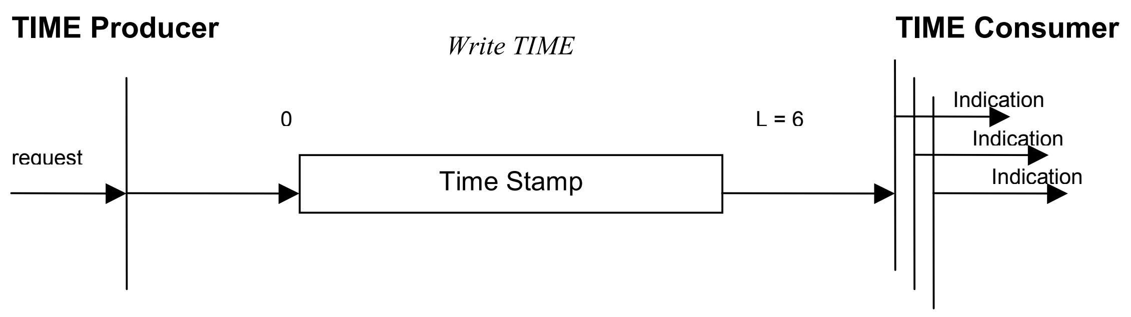


Figure 28 - TIME Protocol

- **Time Stamp:** 6 bytes of the Time Stamp Object

5.1.6 Emergency object (EMCY)

5.1.6.1 Emergency object usage

Emergency objects are triggered by the occurrence of a device internal error situation and are transmitted from an emergency producer on the device. Emergency objects are suitable for interrupt type error alerts. An emergency object shall be transmitted only once per 'error event'. As long as no new errors occur on a device no further emergency objects shall be transmitted.

The emergency object may be received by zero or more emergency consumers. The reaction of the emergency consumer(s) is not specified and does not fall in the scope of this European Standard.

The emergency error codes (Table 20) and the error register (Table B.13) should be used. Device specific additional information and the emergency condition does not fall into the scope of this European Standard.

Table 20 - Emergency error codes

Error Code (hex)	Meaning
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	Current, device input side
22xx	Current inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains Voltage
32xx	Voltage inside the device
33xx	Output Voltage
40xx	Temperature
41xx	Ambient Temperature
42xx	Device Temperature
50xx	Device Hardware
60xx	Device Software
61xx	Internal Software
62xx	User Software
63xx	Data Set
70xx	Additional Modules
80xx	Monitoring
81xx	Communication
8110	CAN Overrun (Objects lost)
8120	CAN in Error Passive Mode
8130	Life Guard Error or Heartbeat Error
8140	recovered from bus off
8150	Transmit COB-ID collision
82xx	Protocol Error
8210	PDO not processed due to length error
8220	PDO length exceeded
90xx	External Error
F0xx	Additional Functions
FFxx	Device specific

The emergency object may be implemented. If a device supports the emergency object, it shall support at least the two error codes 00xx and 10xx.

A device may be in one of two emergency states (Figure 29). Dependent on the transitions emergency objects shall be transmitted. Links between the error state machine and the NMT state machine are defined in the device profiles.

0. After initialisation the device shall enter the error free state if no error is detected. No error message shall be sent.
1. If the device detects an internal error indicated in the first three bytes of the emergency message (error code and error register), it shall enter the error state. An emergency object with the appropriate error code and error register shall be transmitted. The error code shall be filled in at the location of object 1003H (pre-defined error field).
2. If one, but not all error reasons are gone, the emergency message containing error code 0000 (Error reset) may be transmitted together with the remaining errors in the error register and in the manufacturer specific error field.

3. If a new error occurs on the device, it remains in error state and transmits an emergency object with the appropriate error code. The new error code shall be filled in at the top of the array of error codes (1003H). It shall be guaranteed that the error codes are sorted in a timely manner (oldest error - highest sub-index, see Object 1003H).
4. If all errors are repaired, the device shall enter the error free state and shall transmit an emergency object with the error code 'reset error / no error'.

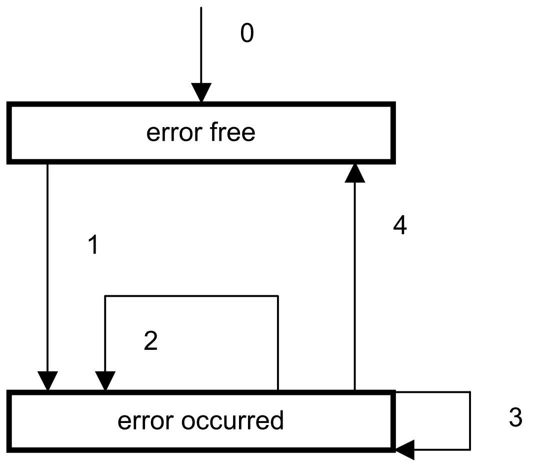


Figure 29 - Emergency State Transition Diagram

5.1.6.2 Emergency object data

The Emergency Telegram shall consist of 8 bytes with the data as shown in Figure 30.

Byte	0	1	2	3	4	5	6	7
Content	Emergency Error Code (see Table 20)		Error register (Object 1001H)	Manufacturer specific Error Field				

Figure 30 - Emergency object data

5.1.6.3 Emergency object services

Emergency object transmission shall follow the producer/consumer push model as described in 4.4.4.

The following object attributes shall be used for emergency objects.

• user type:	notifying device: producer receiving devices: consumer
• data type:	structure of UNSIGNED(16) emergency_error_code, UNSIGNED(8) error_register, ARRAY (5) of UNSIGNED(8) manufacturer_specific_error_field
• inhibit time:	Application specific

5.1.6.4 Emergency object protocol

One unconfirmed service (Write EMCY) is defined (see Figure 31).

Write EMCY

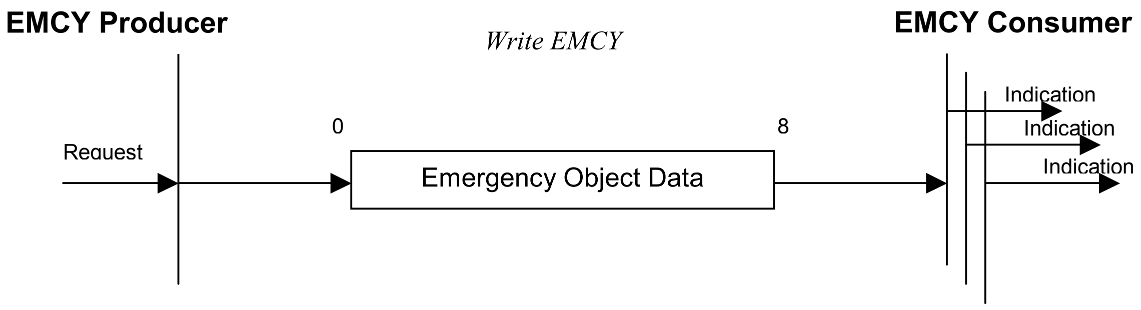


Figure 31 - Emergency object protocol

An emergency object shall not be requested by a remote transmission request (RTR).

5.1.7 Network management objects

5.1.7.1 General

The Network Management (NMT) is node oriented and follows a master-slave structure. NMT objects are used for executing NMT services. Through NMT services, nodes are initialised, started, monitored, reset or stopped. All nodes are regarded as NMT slaves. An NMT Slave shall be uniquely identified in the network by its Node ID, a value in the range of [1..127].

NMT requires that one device in the network shall fulfil the function of the NMT Master.

5.1.7.2 NMT services

5.1.7.2.1 Module control services

Through Module Control Services, the NMT master controls the state of the NMT slaves. The state attribute shall be one of the values {STOPPED, PRE-OPERATIONAL, OPERATIONAL, INITIALISING} (see clause 5.3.2.2 and subclauses). The Module Control Services may be performed with a certain node or with all nodes simultaneously. The NMT master controls its own NMT state machine via local services, which are implementation dependent. The Module Control Services except Start Remote Node may be initiated by the local application.

Start Remote Node

Through this service (see Table 21) the NMT Master sets the state of the selected NMT Slaves to OPERATIONAL.

Table 21 - Start Remote Node

Parameter	*Indication/Request*
Argument	**Mandatory**
Node_ID	selection
All	selection

The service is unconfirmed and shall be implemented. After completion of the service, the state of the selected remote nodes shall be OPERATIONAL.

Stop Remote Node

Through this service (see Table 22) the NMT Master sets the state of the selected NMT Slaves to STOPPED.

Table 22 - Stop Remote Node

Parameter	*Request/Indication*
Argument	**Mandatory**
Node _ID	selection
All	selection

The service is unconfirmed and shall be implemented. After completion of the service, the state of the selected remote nodes shall be STOPPED.

Enter Pre-Operational

Through this service (see Table 23) the NMT Master sets the state of the selected NMT Slave(s) to "PRE-OPERATIONAL".

Table 23 - Enter Pre-Operational

Parameter	Request/Indication
Argument	**Mandatory**
Node-ID	selection
All	selection

The service is unconfirmed and shall be implemented for all devices. After completion of the service, the state of the selected remote nodes shall be PRE-OPERATIONAL.

Reset Node

Through this service (see Table 24) the NMT Master sets the state of the selected NMT Slave(s) from any state to the "reset application" sub-state.

Table 24 - Reset Node

Parameter	Request/Indication
Argument	**Mandatory**
Node-ID	selection
All	selection

The service is unconfirmed and shall be implemented for all devices. After completion of the service, the state of the selected remote nodes shall be RESET APPLICATION.

Reset Communication

Through this service (see Table 25) the NMT Master sets the state of the selected NMT Slave(s) from any state to the "reset communication" sub-state. After completion of the service, the state of the selected remote nodes shall be RESET COMMUNICATION.

Table 25 - Reset Communication

Parameter	Request/Indication
Argument	**Mandatory**
Node-ID	selection
All	selection

The service is unconfirmed and shall be implemented for all devices.

5.1.7.2.2 Error control services

Through Error control services the NMT detects failures in a CAN-based Network.

Local errors in a node may e.g. lead to a reset or change of state. The definition of these local errors does not fall into the scope of this European Standard.

Error Control services are achieved principally through periodically transmitting of messages by a device. There exist two possibilities to perform Error Control.

The guarding is achieved through transmitting guarding requests (Node guarding protocol) by the NMT Master. If an NMT Slave has not responded within a defined span of time (node life time) or if the NMT Slave’s communication status has changed, the NMT Master shall inform its NMT Master Application about that event.

If Life guarding (NMT slave guarded NMT master) is supported, the slave shall use the guard time and life-time factor from its Object dictionary to determine the node life time. If the NMT Slave is not guarded within its life time, the NMT Slave informs its local Application about that event. If guard time and life time factor are 0 (default values), the NMT Slave does not guard the NMT Master.

Guarding shall start for the slave when the first remote-transmit-request for its guarding identifier is received. This may be during the boot-up phase or later.

The heartbeat mechanism for a device is established through cyclically transmitting a message by a heartbeat producer. One or more devices in the network are aware of this heartbeat message. If the heartbeat cycle fails for the heartbeat producer the local application on the heartbeat consumer shall be informed about that event.

Either guarding or heartbeat shall be implemented.

Node Guarding Event

Through this service (see Table 26), the NMT service provider on the NMT Master indicates that a remote error occurred or has been resolved for the remote node identified by Node_ID.

Table 26 - Node Guarding Event

Parameter	*Indication*
Argument	**Mandatory**
Node_ID	mandatory
State	mandatory
Occurred	selection
Resolved	selection

The service is provider initiated and may be implemented.

Life Guarding Event

Through this service (see Table 27), the NMT service provider on an NMT Slave indicates that a remote error occurred or has been resolved.

Table 27 - Life Guarding Event

Parameter	*Indication*
Argument	**Mandatory**
State	mandatory
Occurred	selection
Resolved	selection

The service is provider initiated and may be implemented.

Heartbeat Event

Through this service (see Table 28), the Heartbeat consumer indicates that a heartbeat error occurred or has been resolved for the node identified by Node_ID.

Table 28 - Heartbeat Event

Parameter	*Indication*
Argument	**Mandatory**
Node_ID	mandatory
State	mandatory
Occurred	selection
Resolved	selection

The service is consumer initiated and may be implemented.

5.1.7.2.3 Bootup Service

Bootup Event

Through this service (see Table 29), the NMT slave indicates that a local state transition occurred from the state INITIALISING to the state PRE-OPERATIONAL.

Table 29 - Bootup Event

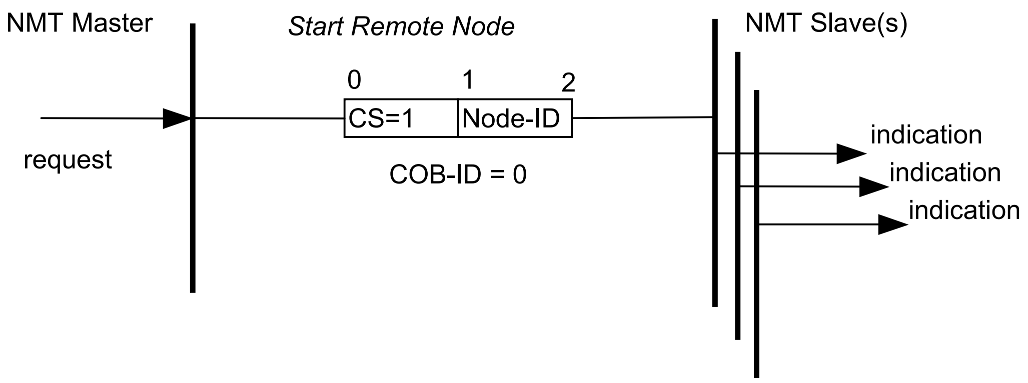
Parameter	*Indication*
Argument	**Mandatory**
None	

The service is provider initiated and shall be implemented.

5.1.7.3 NMT Protocols

5.1.7.3.1 Module Control Protocols

Start Remote Node Protocol



This protocol shall be used to implement the 'Start Remote Node' service (see Figure 32).

Figure 32 - Start Remote Node Protocol

- **cs:** NMT command specifier

  1: start

Stop Remote Node Protocol

This protocol shall be used to implement the 'Stop Remote Node' service (see Figure 33).

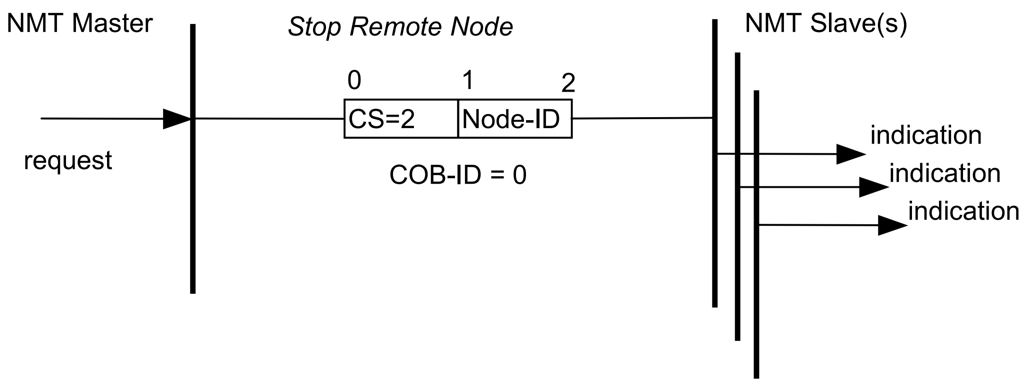
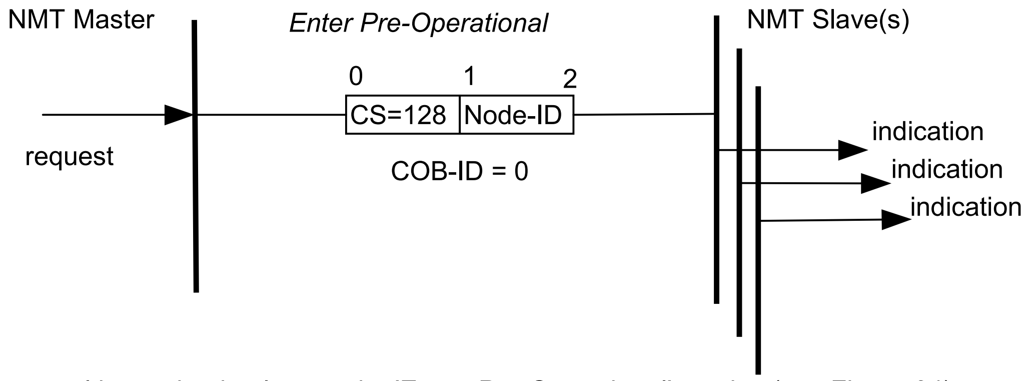


Figure 33 - Stop Remote Node Protocol

cs: NMT command specifier

2: stop

Enter Pre-Operational Protocol



The protocol is used to implement the 'Enter_Pre-Operational' service (see Figure 34).

Figure 34 - Enter Pre-Operational Protocol

- **cs:** NMT command specifier

128:	enter PRE-OPERATIONAL

Reset Node Protocol

The protocol is used to implement the 'Reset Node' service (see Figure 35).

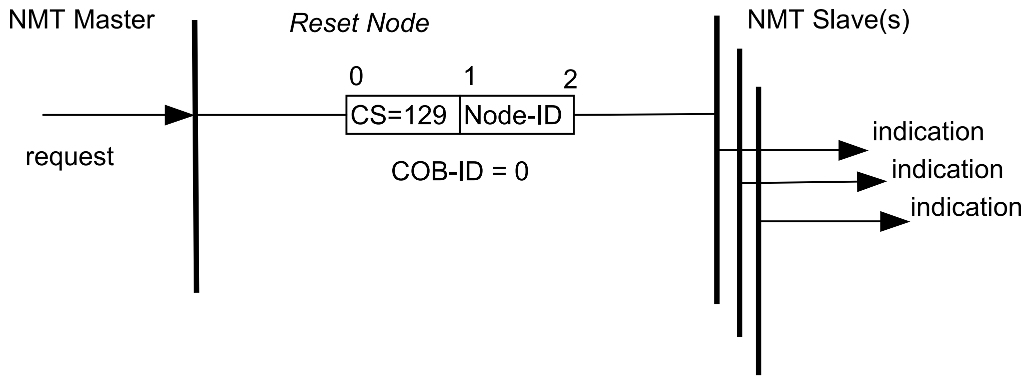


Figure 35 - Reset Node Protocol

cs: NMT command specifier

129:	Reset_Node

Reset Communication Protocol

The protocol is used to implement the 'Reset Communication' service (see Figure 36).

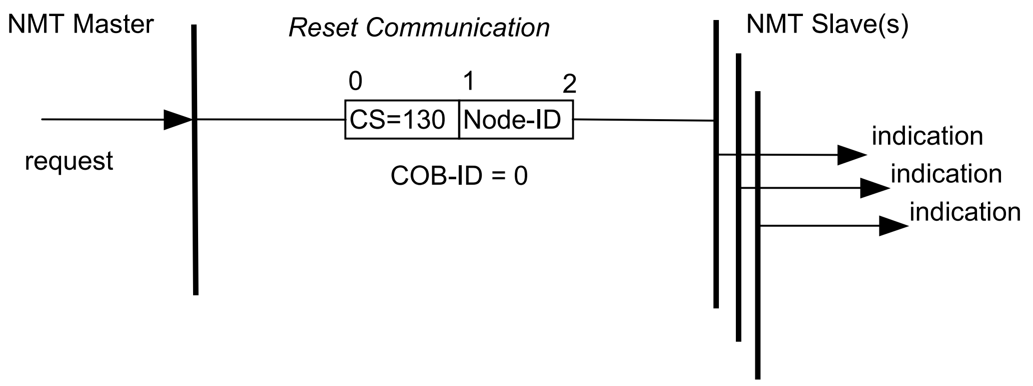


Figure 36 - Reset Communication Protocol

cs: NMT command specifier

130:	Reset_Communication

5.1.7.3.2 Error Control Protocols

Node Guarding Protocol

This protocol is used to detect remote errors in the network (see figure 37). Each NMT Slave shall use one remote COB for the Node Guarding Protocol. This protocol shall implement the provider initiated Error Control services (see Figure 37).

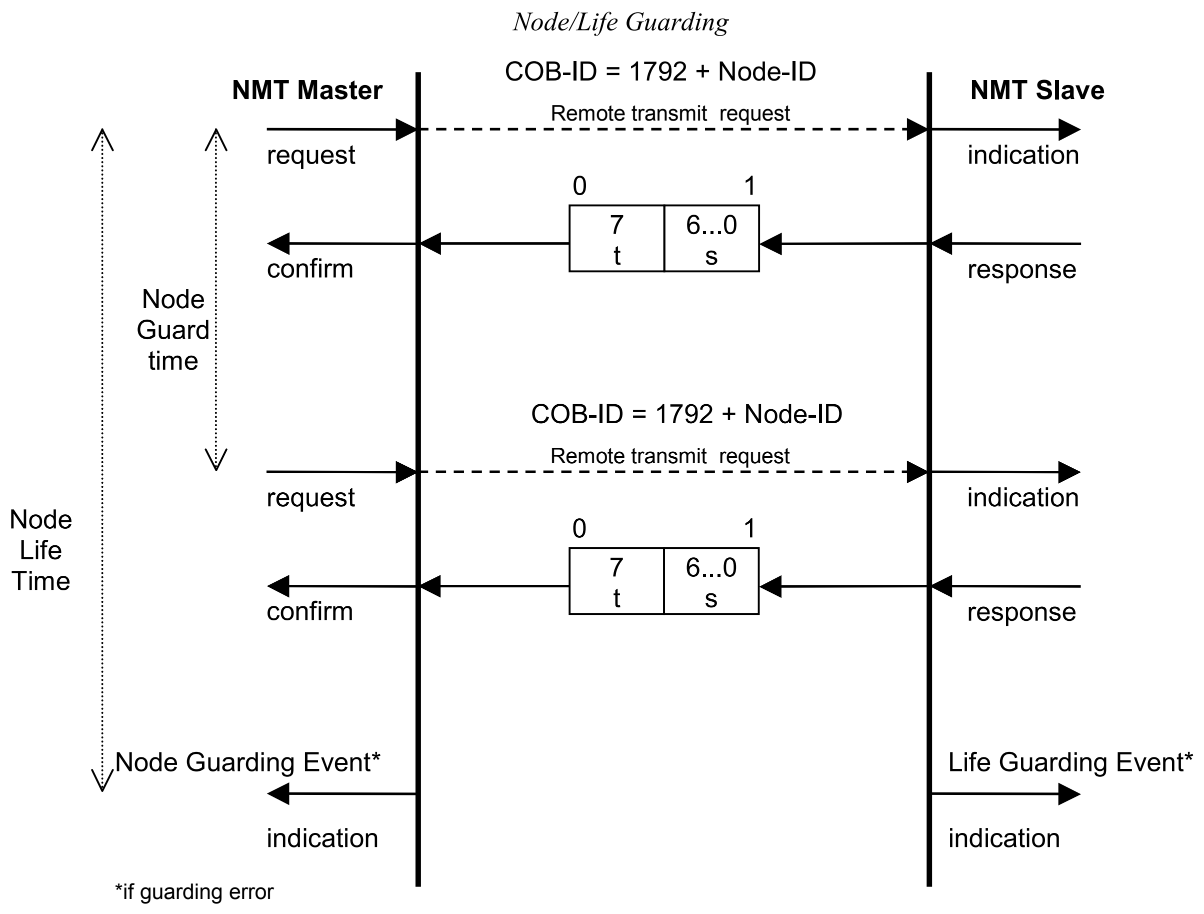


Figure 37 - Node Guarding Protocol

s: the state of the NMT Slave

4:	STOPPED
5:	OPERATIONAL
127:	PRE-OPERATIONAL

t: toggle bit. The value of this bit shall alternate between two consecutive responses from the NMT Slave. If the value of the toggle-bit of the first response after the Guarding Protocol becomes active, it shall be 0. The Toggle Bit in the guarding protocol shall be only reset to 0 when reset_communication is passed (no other change of state resets the toggle bit). If a response is received with the same value of the toggle-bit as in the preceding response then the new response shall be handled as if it was not received.

The NMT Master polls each NMT Slave at regular time intervals. This time-interval is called the guard time and may be different for each NMT Slave. The response of the NMT Slave shall contain the state of that NMT Slave. The node life time is given by the guard time multiplied by the life time factor. The node life time may be different for each NMT Slave. If the NMT Slave has not been polled during its life time, a remote node error shall be indicated through the 'Life Guarding Event' service.

A remote node error shall be indicated through the 'Node guarding event' service if

- the remote transmit request is not confirmed within the node life time,
- the reported NMT slave state does not match the expected state.

If it has been indicated that a remote error has occurred and the errors in the guarding protocol have disappeared, it shall be indicated that the remote error has been resolved through the 'Node Guarding Event' and 'Life Guarding Event' services.

For the guard time and the life time factor there are default values specified at the appropriate Object dictionary entries.

Heartbeat Protocol

The Heartbeat Protocol (see Figure 38) defines an Error Control Service without need for remote frames. A Heartbeat Producer shall transmit a Heartbeat message cyclically. One or more Heartbeat Consumer receive the indication. The relationship between producer and consumer is configurable via the object dictionary. The Heartbeat Consumer shall guard the reception of the Heartbeat within the Heartbeat Consumer Time. If the Heartbeat is not received within the Heartbeat Consumer Time a Heartbeat Event shall be generated.

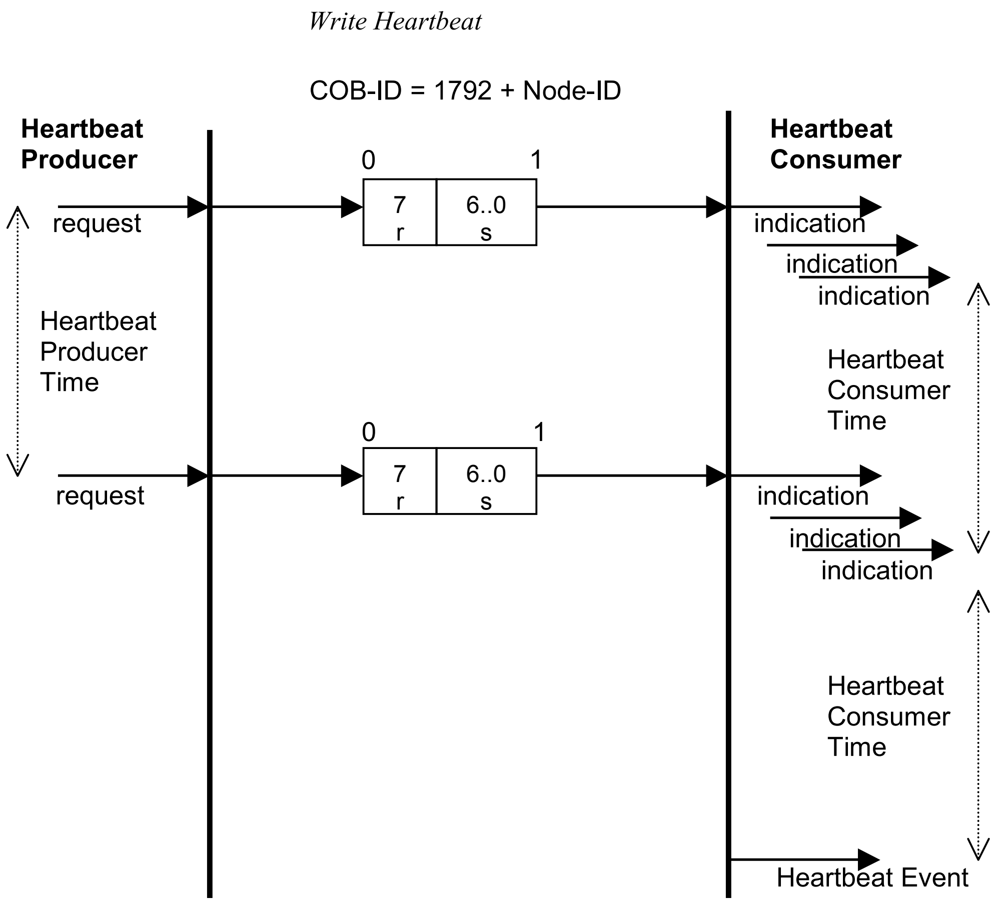


Figure 38 - Heartbeat Protocol

- **r:** reserved (always 0)
- **s:** the state of the Heartbeat producer

0:	BOOTUP
4:	STOPPED
5:	OPERATIONAL
127:	PRE-OPERATIONAL

If the Heartbeat Producer Time is configured on a device, the Heartbeat Protocol shall begin immediately. If a device starts with a value for the Heartbeat Producer Time unequal to 0, the Heartbeat Protocol shall start on the state transition from INITIALISING to PRE-OPERATIONAL. In this case the Boot-up Message is regarded as first heartbeat message. A device shall not use both error control mechanisms Guarding Protocol and Heartbeat Protocol at the same time. If the heartbeat producer time is unequal 0 the heartbeat protocol shall be used.

5.1.7.3.3 Boot-up Protocol

This protocol shall be used to signal that a NMT slave has entered the PRE-OPERATIONAL state after INITIALISING (see Figure 39).

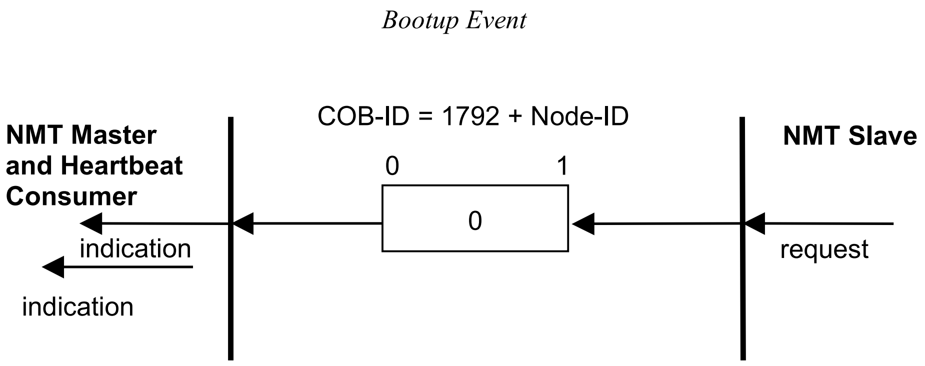


Figure 39 - Boot-up Protocol

The boot-up protocol shall use the very same COB-ID as the error control protocol. One data byte shall be transmitted with value 0.

5.2 Synchronisation of the SYNC Consumer

5.2.1 Transmission of Synchronous PDO Messages

Synchronous transmission of a message means that the transmission of the message is fixed in time with respect to the transmission of the SYNC message. The synchronous message is transmitted within a given time window with respect to the SYNC transmission, and at most once for every period of the SYNC.

In general the fixing of the transmission time of synchronous PDO messages coupled with the periodicity of transmission of the SYNC message guarantees that devices can sample process variables from a process environment and apply their actuation in a co-ordinated fashion.

A device consuming SYNC messages will provide synchronous PDO messages too. The reception of a SYNC message controls the moment when the application will interact with the process environment according to the contents of a synchronous PDO. The synchronous mechanism is intended for transferring manipulated values and actual values on a fixed timely base (see Figures 40 and 41).

In general a synchronous PDO with a manipulated value will be received before a SYNC. The SYNC consuming device will actuate based on this synchronous PDO at the next SYNC message. The reception of a SYNC will also prompt a device operating in the cyclic mode to sample its feedback data and transmit a synchronous PDO with an actual value as soon as possible afterwards.

Depending upon its capabilities, a device may also be parameterised with the time period *synchronous window length* after the SYNC at which it is guaranteed that its commanded value has arrived. It can therefore perform any processing on the manipulated value which is required in order to actuate at the next SYNC message.

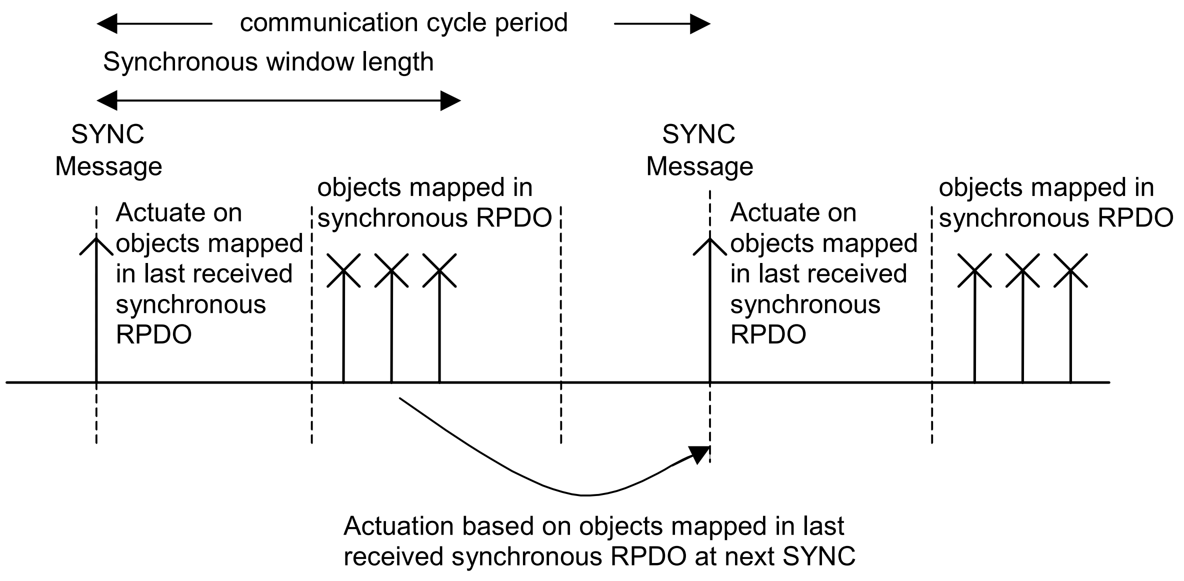

communication cycle period
Synchronous window length
SYNC Message
Actuate on objects mapped in last received synchronous RPDO
objects mapped in synchronous RPDO
SYNC Message
Actuate on objects mapped in last received synchronous RPDO
objects mapped in synchronous RPDO
Actuation based on objects mapped in last received synchronous RPDO at next SYNC

Figure 40 - Bus Synchronisation and Actuation

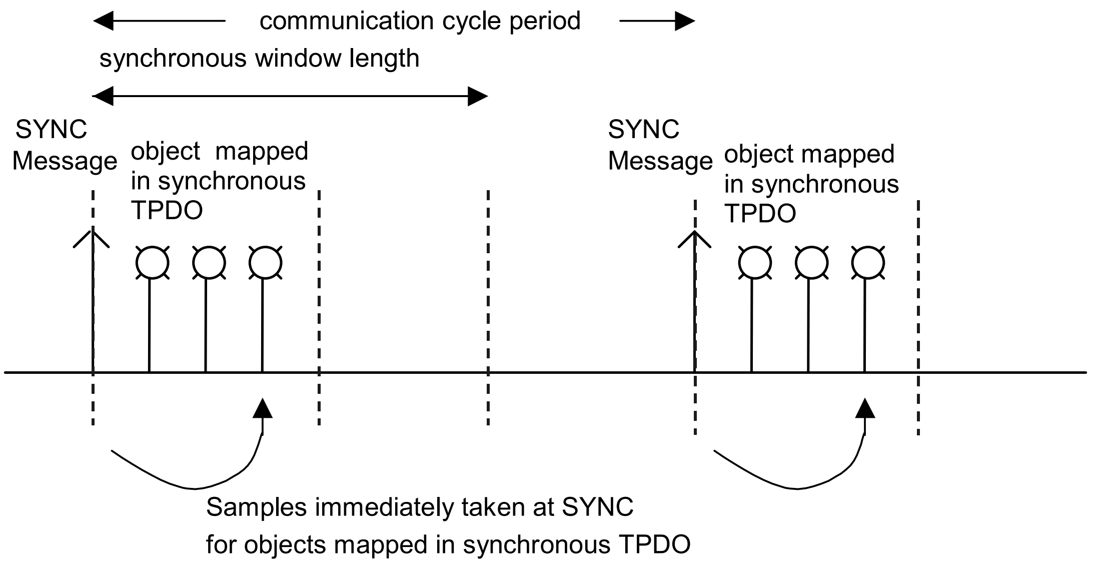

communication cycle period
synchronous window length
SYNC Message
object mapped in synchronous TPDO
SYNC Message
object mapped in synchronous TPDO
Samples immediately taken at SYNC for objects mapped in synchronous TPDO

Figure 41 - Bus Synchronisation and Sampling

5.2.2 Optional High Resolution Synchronisation Protocol

The synchronisation message shall carry no data and is therefore easy to generate. However, the jitter of this SYNC depends on the bit rate of the bus as even the very high priority SYNC shall wait for the current message on the bus to be transmitted before it gains bus access.

Some time critical applications especially in large networks with reduced transmission rates require more accurate synchronisation; it may be necessary to synchronise the local clocks with an accuracy in the order of microseconds. This is achieved by using the optional high resolution synchronisation protocol which employs a special form of time stamp message (see Figure 42) to adjust the inevitable drift of the local clocks.

The SYNC producer time-stamps the interrupt generated at t1 by the successful transmission of the SYNC message (this takes until t2). After that (at t4) he sends a time-stamp message containing the corrected time-stamp (t1) for the SYNC transmission success indication. The SYNC consumer that have taken local time-stamps (t3) on the reception (t1) of the SYNC may now compare their corrected time-stamp (t1) with the one received in the time-stamp message from the SYNC producer. The difference between these values determines the amount of time to adjust the local clock.

With this protocol only the local latencies (t2-t1 on the SYNC producer and t3-t1 on the SYNC consumer) are time critical. These latencies depend on local parameters (like interrupt processing times and hardware delays) on the nodes which have to be determined once. The accuracy of this determination is implementation specific, it forms the limiting factor of the synchronisation (or clock adjustment) accuracy. Note that each node only shall know its own latency time as the time-stamp message contains the corrected value t1 and not t2.

The time-stamp shall be encoded as UNSIGNED32 with a resolution of 1 microsecond which means that the time counter restarts every 72 min. It is configured by mapping the high resolution time-stamp into a PDO.

It is reasonable to repeat the clock adjustment only when the maximum drift of the local clock exceeds the synchronisation accuracy. For most implementations this means that it is sufficient to add this time-stamp message to the standard SYNC once every second.

This principle enables the best accuracy that may be achieved with bus-based synchronisation, especially when implemented on CAN controllers that support time-stamping. Note that the accuracy is widely independent of the transmission rate. Further improvement requires separate hardware (e.g. wiring).

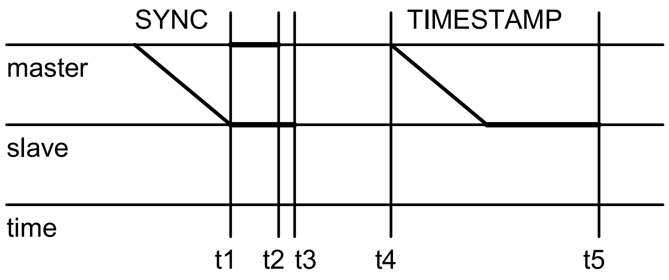


Figure 42 - Optional High Resolution Synchronisation Protocol

5.3 Network Initialisation and System Boot-Up

5.3.1 Initialisation Procedure

In Figure 43 the general flow chart of the network initialisation process, controlled by a NMT Master application or configuration application is shown.

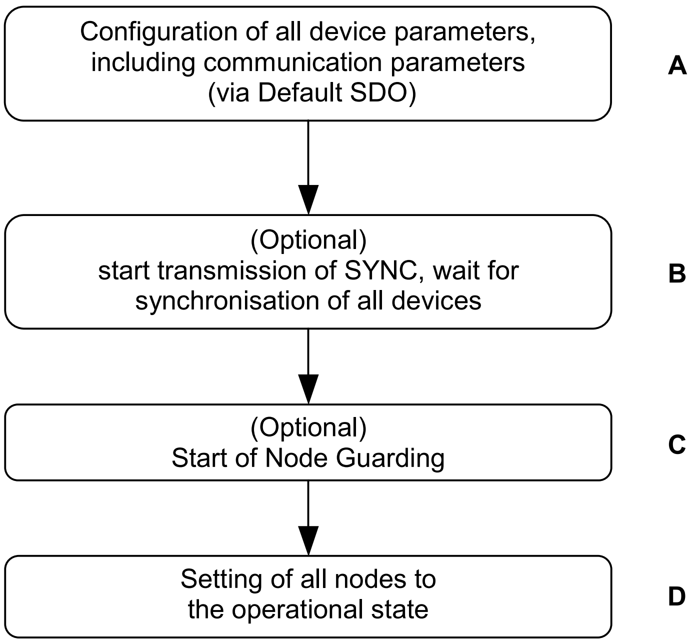

Configuration of all device parameters, including communication parameters (via Default SDO)
A
(Optional) start transmission of SYNC, wait for synchronisation of all devices
B
(Optional) Start of Node Guarding
C
Setting of all nodes to the operational state
D

Figure 43 - Flow Chart of the Network Initialisation Process

In step A the devices are in the node state PRE-OPERATIONAL which shall be entered automatically after power-on. In this state the devices are accessible via their Default-SDO using identifiers that have been assigned according to the Predefined Connection Set. In this step the configuration of device parameters takes place on all nodes which support parameter configuration.

This is done from a Configuration Application or Tool which resides on the node that is the client for the default SDOs. For devices that support these features the selection and/or configuration of PDOs, the mapping of application objects (PDO mapping), the configuration of additional SDOs and the setting of COB-IDs may be performed via the Default-SDO objects.

In many cases a configuration is not even necessary as default values are defined for all application and communication parameters.

If the application requires the synchronisation of all or some nodes in the network, the appropriate mechanisms may be initiated in the Step B. It may be used to ensure that all nodes are synchronised by the SYNC object before entering the node state OPERATIONAL in step D. The first transmission of SYNC object starts within 1 sync cycle after entering the PRE-OPERATIONAL state.

In Step C Node guarding may be activated (if supported) using the guarding parameters configured in step A.

With step D all nodes are enabled to communicate via their PDO objects.

5.3.2 NMT State Machine

5.3.2.1 Overview

In Figure 44 the state diagram of a device is shown. Devices shall enter the PRE-OPERATIONAL state directly after finishing the device initialisation. During this state device parameterisation and ID-allocation via SDO (e.g. using a configuration tool) is possible. Then the nodes may be switched directly into the OPERATIONAL state.

The NMT state machine determines the behaviour of the Communication function unit (see 4.3). The coupling of the application state machine to the NMT state machine is device dependent and falls into the scope of device profiles.

Minimal Boot-up shall consist of one CAN telegram: a broadcast Start_Remote_Node message.

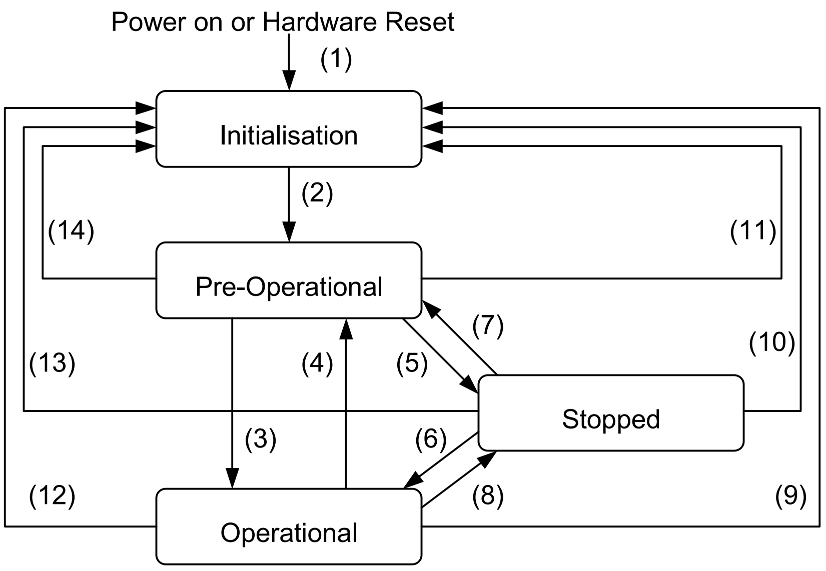


Figure 44 - State Diagram of a Device

Table 30 defines the trigger for state transitions.

Table 30 - Trigger for State Transition

(1)	At Power on the initialisation state is entered autonomously
(2)	Initialisation finished - enter PRE-OPERATIONAL automatically
(3),(6)	Start_Remote_Node indication
(4),(7)	Enter_PRE-OPERATIONAL_State indication
(5),(8)	Stop_Remote_Node indication
(9),(10),(11)	Reset_Node indication
(12),(13),(14)	Reset_Communication indication

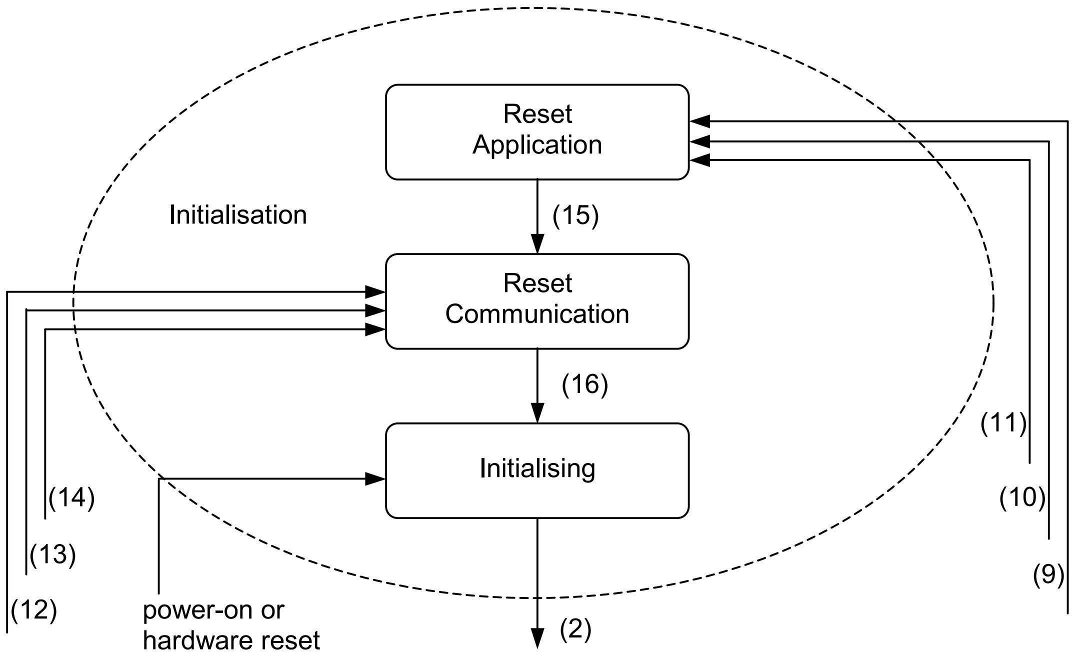
5.3.2.2 States

5.3.2.2.1 Initialisation

The „initialisation“ state is divided into three sub-states (see Figure 45) in order to enable a complete or partial reset of a node.

1. **Reset_Application**: In this state the parameters of the manufacturer specific profile area and of the standardised device profile area shall be set to their power-on values. After setting of the power-on values the state Reset_Communication shall be entered autonomously.
2. **Reset_Communication**: In this state the parameters of the communication profile area shall be set to their power-on values. After this the state Initialising shall be entered autonomously.
3. **Initialising**: This is the first sub-state the device enters after power-on or hardware reset. After finishing the basic node initialisation the device executes the write boot-up object service and shall enter the state PRE-OPERATIONAL autonomously.

Power-on values are the last stored parameters. If storing is not supported or has not been executed or if the reset was preceded by a restore_default command (object 1011H), the power-on values shall be the default values according to the communication and device profile specifications.



(2)	Initialisation finished - enter PRE-OPERATIONAL automatically
(12),(13),(14)	Reset_Communication indication
(9),(10)(11)	Reset_Node indication
(15)	Application Reset performed
(16)	Communication Reset performed

Figure 45 - Structure of the Initialisation state

5.3.2.2.2 Pre-Operational

In the PRE-OPERATIONAL state, communication via SDOs shall be supported. PDOs shall not exist, so PDO communication shall not be allowed. Configuration of PDOs, device parameters and also the allocation of application objects (PDO-mapping) may be performed by a configuration application.

The node shall be switched into the operational state directly by sending a Start_Remote_Node request.

5.3.2.2.3 Operational

In the OPERATIONAL state all communication objects shall be active. Transitioning to OPERATIONAL shall create all PDOs; the constructor uses the parameters as described in the Object Dictionary. Object Dictionary Access via SDO shall be supported. Implementation aspects or the application state machine however may require to limit the access to certain objects whilst being operational, e.g. an object may contain the application program which may not be changed during execution.

5.3.2.2.4 Stopped

By switching a device into the Stopped state it shall be forced to stop the communication altogether (except node guarding or heartbeat). Furthermore, this state may be used to achieve certain application behaviour. The definition of this behaviour falls into the scope of device profiles.

5.3.2.3 States and Communication Object Relation

Table 31 shows the relation between communication states and communication objects. Services on the listed communication objects shall only be executed if the devices involved in the communication are in the appropriate communication states.

Table 31 - States and Communication Objects

	INITIALISING	PRE-OPERATIONAL	OPERATIONAL	STOPPED
PDO			X	
SDO		X	X	
Synchronisation Object		X	X	
Time Stamp Object		X	X	
Emergency Object		X	X	
Boot-Up Object	X			
Network Management Objects		X	X	X

5.3.2.4 State Transitions

State transitions shall be caused by

- reception of an NMT object used for module control services,
- hardware reset,
- Module Control Services locally initiated by application events, defined by device profiles.

5.3.3 Pre-Defined Connection Set

In order to reduce configuration effort for simple networks a default identifier allocation scheme is defined. These identifiers shall be available in the PRE-OPERATIONAL state directly after initialisation (if no modifications have been stored). The objects SYNC, TIME STAMP, EMERGENCY and PDOs may be deleted and re-created with new identifiers by means of dynamic distribution. A device shall provide the corresponding identifiers only for the supported communication objects.

The default profile ID-allocation scheme (Figure 46) consists of a functional part, which determines the object priority and a Node-ID part, which allows to distinguish between devices of the same functionality. This allows a peer-to-peer communication between a single master device and up to 127 slave devices. It also supports the broadcasting of non-confirmed NMT-objects, SYNC- and TIME-STAMP-objects. Broadcasting shall be indicated by a Node-ID of zero.

The pre-defined connection set supports one emergency object, one SDO, up to 4 Receive-PDOs (RPDO) and up to 4 Transmit-PDOs (TPDO) as well as the NMT object and NMT error control object.

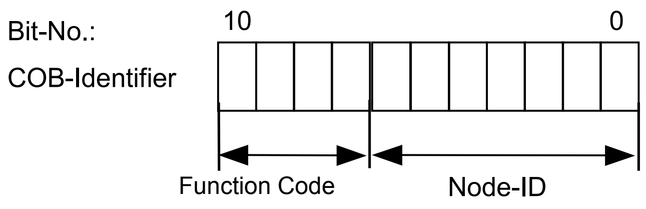


Figure 46 - Identifier allocation scheme for the pre-defined connection set

Table 32 and Table 33 show the supported objects and their allocated COB-IDs.

Table 32 - Broadcast Objects of the Pre-defined Connection Set

Object	Function code (binary)	Resulting COB-ID	Communication parameters at Index
NMT	0000	0	-
SYNC	0001	128 (80h)	1005h, 1006h, 1007h
TIME STAMP	0010	256 (100h)	1012h, 1013h

Table 33 - Peer-to-Peer Objects of the Pre-defined Connection Set

Object	Function code (binary)	Resulting COB-IDs	Communication parameters at Index
EMERGENCY	0001	129 (81h) – 255 (FFh)	1014h, 1015h
PDO1 (tx)	0011	385 (181h) – 511 (1FFh)	1800h
PDO1 (rx)	0100	513 (201h) – 639 (27Fh)	1400h
PDO2 (tx)	0101	641 (281h) – 767 (2FFh)	1801h
PDO2 (rx)	0110	769 (301h) – 895 (37Fh)	1401h
PDO3 (tx)	0111	897 (381h) – 1023 (3FFh)	1802h
PDO3 (rx)	1000	1025 (401h) – 1151 (47Fh)	1402h
PDO4 (tx)	1001	1153 (481h) – 1279 (4FFh)	1803h
PDO4 (rx)	1010	1281 (501h) – 1407 (57Fh)	1403h
SDO (tx)	1011	1409 (581h) – 1535 (5FFh)	1200h
SDO (rx)	1100	1537 (601h) – 1663 (67Fh)	1200h
NMT Error Control	1110	1793 (701h) – 1919 (77Fh)	1016h, 1017h

Table 33 shall be seen from the devices point of view.

The pre-defined connection set shall always apply to the standard CAN frame with 11-bit Identifier, even if extended CAN frames are present in the network.

6 Product information

6.1 Instructions for installation, operation and maintenance

These shall be in accordance with 6.1 of EN 50325-1.

6.2 Marking

These shall be in accordance with 6.2 of EN 50325-1.

7 Normal service, transport and mounting conditions

7.1 Normal service conditions

7.1.1 General

Devices of a CANopen network shall be capable of operating under the conditions as specified in EN 50325-1. Additional requirements may be given by application-specific standards.

7.1.2 Ambient air temperature

These shall be in accordance with 7.1.2 of EN 50325-1.

7.1.3 Altitude

These shall be in accordance with 7.1.3 of EN 50325-1.

7.1.4 Humidity

These shall be in accordance with 7.1.4 of EN 50325-1.

7.1.5 Pollution degree

These shall be in accordance with 7.1.5 of EN 50325-1.

7.1.6 Sealed connectors

CANopen devices may be provided with sealed connectors in accordance with 7.1.6 of EN 50325-1.

7.2 Conditions during transport and storage

These shall be in accordance with 7.2 of EN 50325-1.

7.3 Mounting

These shall be in accordance with 7.3 of EN 50325-1.

8 Constructional and performance requirements

8.1 General

There are no power supply, cable, or connector specified, because EN 50325-4 defines only the communication interface. However, the general physical layer specifications as defined in ISO 11898 shall be met.

8.2 Physical layer

8.2.1 General

The physical medium for devices shall be a differentially driven two-wire bus line with common return according to high-speed transmission specification in ISO 11898.

8.2.2 Transceiver

Using the high-speed transceiver according to ISO 11898 the maximum rating for V_{CAN_H} and V_{CAN_L} shall be +16V. Galvanic isolation between bus nodes may be provided. It is recommended to use a transceiver that is capable of sustaining disconnection of any of the wires of the connector including the optional V+ voltages of up to 30V.

8.2.3 Bit rates and timing

The recommended bit rates and corresponding bit timings are listed in Table 34. One of these bit rates shall be supported.

Table 34 - Recommended Bit Timing Settings

Bit rate Bus length [(1)]	Nominal bit time t_b	Number of time quanta per bit	Length of time quantum t_q	Location of sample point
1 Mbit/s 25 m	1 µs	8	125 ns	6 t_q (750 ns)
800 kbit/s 50 m	1,25 µs	10	125 ns	8 t_q (1 µs)
500 kbit/s 100 m	2 µs	16	125 ns	14 t_q (1,75 µs)
250 kbit/s 250 m [(2)]	4 µs	16	250 ns	14 t_q (3,5 µs)
125 kbit/s 500 m [(2)]	8 µs	16	500 ns	14 t_q (7 µs)
50 kbit/s 1000 m [(3)]	20 µs	16	1,25 µs	14 t_q (17,5 µs)
20 kbit/s 2500 m [(3)]	50 µs	16	3,125 µs	14 t_q (43,75 µs)
10 kbit/s 5000 m [(3)]	100 µs	16	6,25 µs	14 t_q (87,5 µs)

The bit timing settings are examples based on the following assumptions:

- oscillator frequency	16 MHz +/-0,1%;	
- sampling mode	Single sampling	SAM = 0;
- synchronisation mode	Recessive to dominant edges only	SYNC = 0;
- synchronisation jump width	1 * t_q	SJW = 0;
- phase Segment 2	2 * t_q	TSEG2 = 1.

Note 1 Rounded bus length estimation (worst case) on basis 5 ns/m propagation delay and a total effective device internal in-out delay as follows:

- 1Mbit/s - 800 kbit/s:	210 ns;
- 500 kbit/s - 250 kbit/s:	300 ns (includes 2 * 40 ns for optocouplers);
- 125 kbit/s:	450 ns (includes 2 * 100 ns for optocouplers);
- 50 kbit/s - 10 kbit/s:	1,5 t_q; Effective delay = delay recessive to dominant plus dominant to recessive divided by two.

Note 2 For bus length greater than about 200 m optocouplers should be used. If optocouplers are placed between CAN controller and transceiver this affects the maximum bus length depending upon the propagation delay of the optocouplers i.e. -4m per 10 ns propagation delay of employed optocoupler type.

Note 3 For bus length greater than about 1 km bridge or repeater devices may be needed.

Note 4 The bit timings in the table are calculated for an oscillator frequency of 16 MHz. If another oscillator is used the number of time quanta may be different. Nevertheless the location of the sample point shall be as near as possible at the recommended sample point.

8.3 Electromagnetic compatibility (EMC)

Devices compliant to EN 50325-4 shall meet all requirements of the specific application field. For industrial control applications the following standards should be considered:

- EN 50081-2 ;
- EN 61000-6-2 ;
- EN 55011 ;
- EN 61000-4.

9 Tests

All implemented communication services shall be in compliance with this European Standard. Physical layer test circuits are specified in ISO 11898.

9.1 Conformance Testing

Information on conformance testing services are offered by the following institution:

CAN in Automation (CiA) GmbH
Am Weichselgarten 26,
D-91058 Erlangen, Germany
www.can-cia.org.

Annex A
(normative)

Data types and encoding rules

A.1 General description of data types and encoding rules

To be able to exchange meaningful data across the CAN network, producer and consumer(s) shall use the same data format. The encoding rules define the representation of values of data types and the CAN network transfer syntax for the representations. Values are represented as bit sequences. Bit sequences shall be transferred in sequences of octets (bytes). For numerical data types the encoding shall be little endian style as shown in Figure A.1.

Applications often require data types beyond the basic data types. Using the compound data type mechanism the list of available data types may be extended. Some general extended data types are defined as “Visible String” or “Time of Day” for example. The compound data types are a means to implement user defined “DEFTYPES” in the terminology of this specification and not “DEFSTRUCTS” (see Table B.2).

A.2 Data type definitions

A.2.1 General

A data type determines a relation between values and encoding for data of that type. Names are assigned to data types in their type definitions. The syntax of data and data type definitions shall be as follows (see EN 61131-3).

data_definition	::= type_name data_name
type_definition	::= constructor type_name
constructor	::= compound_constructor | basic_constructor
compound_constructor	::= array_constructor | structure_constructor
array_constructor	::= ‘ARRAY’ ‘[‘ length ‘]’ ‘OF’ type_name
structure_constructor	::= ‘STRUCT’ ‘OF’ component_list
component_list	::= component { ‘,’ component }
component	::= type_name component_name
basic_constructor	::= ‘BOOLEAN’ | ‘VOID’ bit_size | ‘INTEGER’ bit_size | ‘UNSIGNED’ bit_size | ‘REAL32’ | ‘REAL64’ | ‘NIL’
bit_size	::= ‘1’ | ‘2’ | <...> | ‘64’
length	::= positive_integer
data_name	::= symbolic_name
type_name	::= symbolic_name
component_name	::= symbolic_name
symbolic_name	::= letter { [‘_’] (letter | digit) }
positive_integer	::= (‘1’ | ‘2’ | <...> | ‘9’) { digit }
letter	::= ‘A’ | ‘B’ | <...> | ‘Z’ | ‘a’ | ‘b’ | <...> | ‘z’
digit	::= ‘0’ | ‘1’ | <...> | ‘9’

Recursive definitions are not allowed.

The data type defined by type_definition is called basic (res.~compound) when the constructor is basic_constructor (res. compound_constructor).

A.2.2 Bit Sequences

A.2.2.1 Definition of Bit Sequences

A bit shall take the values 0 or 1. A bit sequence *b* is an ordered set of 0 or more bits. If a bit sequence *b* contains more than 0 bits, they shall be denoted as b_j, $j \geq 0$. Let $b_0, ..., b_{n-1}$ be bits, *n* a positive integer. Then

$$b = b_0\ b_1\ ...\ b_{n-1}$$

is a bit sequence of length $|b| = n$. The empty bit sequence of length 0 shall be denoted ε.

Examples: 10110100, 1, 101, etc. are bit sequences.

The inversion operator ($\neg$) on bit sequences shall assign to a bit sequence

$$b = b_0\ b_1\ ...\ b_{n-1}$$

the bit sequence

$$\neg b = \neg b_0\ \neg b_1\ ...\ \neg b_{n-1}$$

Here is $\neg 0 = 1$ and $\neg 1 = 0$ on bits.

The basic operation on bit sequences is concatenation.

Let $a = a_0\ ...\ a_{m-1}$ and $b = b_0\ ...\ b_{n-1}$ be bit sequences. Then the concatenation of *a* and *b*, denoted *ab*, shall be

$$ab = a_0\ ...\ a_{m-1}\ b_0\ ...\ b_{n-1}$$

Example: (10)(111) = 10111 is the concatenation of 10 and 111.

The following holds for arbitrary bit sequences *a* and *b*:

$$|ab| = |a| + |b|$$

and

$$\varepsilon a = a\varepsilon = a$$

A.2.2.2 Transfer Syntax for Bit Sequences

For transmission across a CAN network a bit sequence shall be reordered into a sequence of octets. Here and in the following hexadecimal notation is used for octets. Let $b = b_0...b_{n-1}$ be a bit sequence with $n \leq 64$. Denote *k* a non-negative integer such that $8(k - 1) < n \leq 8k$. Then *b* is transferred in *k* octets assembled as shown in Figure A.1. The bits b_i, $i \geq n$ of the highest numbered octet shall be 'do not care' bits.

Octet 1 shall be transmitted first and octet *k* shall be transmitted last. Hence the bit sequence shall be transferred as follows across the CAN network:

$$b_7, b_6, ..., b_0, b_{15}, ..., b_8, ...$$

octet number	1.	2.	k.
	b_7 .. b_0	b_{15} .. b_8	b_{8k-1} .. b_{8k-8}

Figure A.1 - Transfer Syntax for Bit Sequences

Example:

Bit 9	...	Bit 0
10	0001	1100
2h	1h	Ch
		= 21Ch

The bit sequence b = b_0 .. b_9 = 0011 1000 01 represents an UNSIGNED10 with the value 21Ch and is transferred in two octets:

First 1Ch and then 02h.

A.2.3 Basic Data Types

A.2.3.1 General

For basic data types "type_name" shall equal the literal string of the associated constructor (aka *symbolic_name*), e.g.,

BOOLEAN BOOLEAN

is the type definition for the BOOLEAN data type.

A.2.3.2 NIL

Data of basic data type *NIL* shall be represented by ε.

A.2.3.3 Boolean

Data of basic data type *BOOLEAN* shall attain the values TRUE or FALSE. The values shall be represented as bit sequences of length 1. The value TRUE (res. FALSE) shall be represented by the bit sequence 1 (res. 0).

A.2.3.4 Void

Data of basic data type *VOIDn* shall be represented as bit sequences of length *n* bit. The value of data of type VOIDn is undefined. The bits in the sequence of data of type VOIDn shall either be specified explicitly or else marked "do not care".

Data of type VOIDn is useful for reserved fields and for aligning components of compound values on octet boundaries.

A.2.3.5 Unsigned Integer

Data of basic data type *UNSIGNEDn* has values in the non-negative integers. The value range is 0, ..., 2^n-1. The data shall be represented as bit sequences of length *n*. The bit sequence

$$b = b_0 \ldots b_{n-1}$$

shall be assigned the value

$$\text{UNSIGNEDn}(b) = b_{n-1}\, 2^{n-1} + \ldots + b_1\, 2^1 + b_0\, 2^0$$

Note that the bit sequence shall start on the left with the least significant byte.

Example: The value 266 = 10Ah with data type UNSIGNED16 is transferred in two octets across the bus, first 0Ah and then 01h.

The following UNSIGNEDn data types are transferred as shown in Figure A.2.

octet number	**1.**	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**
UNSIGNED8	$b_7..b_0$							
UNSIGNED16	$b_7..b_0$	$b_{15}..b_8$						
UNSIGNED24	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$					
UNSIGNED32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
UNSIGNED40	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$			
UNSIGNED48	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$		
UNSIGNED56	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	
UNSIGNED64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

Figure A.2 - Transfer syntax for data type UNSIGNEDn

A.2.3.6 Signed Integer

Data of basic data type *INTEGERn* shall have values in the integers. The value range is $-2^{n-1}, ..., 2^{n-1}-1$. The data shall be represented as bit sequences of length *n*. The bit sequence

$$b = b_0 .. b_{n-1}$$

shall be assigned the value

$$\text{INTEGERn}(b) = b_{n-2}\, 2^{n-2} + ...+ b_1\, 2^1 + b_0\, 2^0 \quad \text{if } b_{n-1} = 0$$

and, performing two's complement arithmetic,

$$\text{INTEGERn}(b) = -\ \text{INTEGERn}(\text{^}b) - 1 \quad \text{if } b_{n-1} = 1$$

Note that the bit sequence shall start on the left with the least significant bit.

Example: The value –266 = FEF6h with data type INTEGER16 is transferred in two octets across the bus, first F6h and then FEh.

The following INTEGERn data types are transferred as shown in Figure A.3.

octet number	**1.**	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**
INTEGER8	$b_7..b_0$							
INTEGER16	$b_7..b_0$	$b_{15}..b_8$						
INTEGER24	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$					
INTEGER32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
INTEGER40	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$			
INTEGER48	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$		
INTEGER56	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	
INTEGER64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

Figure A.3 - Transfer syntax for data type INTEGERn

A.2.3.6 Floating-Point Numbers

Data of basic data types *REAL32* and *REAL64* shall have values in the real numbers.

The data type *REAL32* shall be represented as bit sequence of length 32. The encoding of values shall follow the IEEE 754-1985 Standard for single precision floating-point.

The data type *REAL64* shall be represented as bit sequence of length 64. The encoding of values shall follow the IEEE 754-1985 Standard for double precision floating-point numbers.

A bit sequence of length 32 either shall have a value (finite non-zero real number, ±0, ± _) or shall be NaN (not-a-number). The bit sequence

$$b = b_0 \dots b_{31}$$

is assigned the value (finite non-zero number)

$$\text{REAL32}(b) = (-1)^S \, 2^{E - 127} (1 + F)$$

Here

$S = b_{31}$ is the sign.

$E = b_{30}\, 2^7 + \dots + b_{23}\, 2^0$, $0 < E < 255$, is the un-biased exponent.

$F = 2^{-23}$ $(b_{22}\, 2^{22} + \dots + b_1\, 2^1 + b_0\, 2^0)$ is the fractional part of the number.

$E = 0$ is used to represent ± 0. $E = 255$ is used to represent infinities and NaN's.

Note that the bit sequence shall start on the left with the least significant bit.

Example:

$6.25 = 2^{E - 127} (1 + F)$ with

$E = 129 = 2^7 + 2^0$ and

$F = 2^{-1} + 2^{-4} = 2^{-23}(2^{22} + 2^{19})$ hence the number is represented as:

S	*E*	*F*
b_{31}	*$b_{30} .. b_{23}$*	*$b_{22} .. b_0$*
0	*100 0000 1*	*100 1000 0000 0000 0000 0000*

$6.25 = b_0 .. b_{31}$ = 0000 0000 0000 0000 0001 0011 0000 0010

It shall be transferred in the order shown in Figure A.4.

octet number	**1.**	**2.**	**3.**	**4.**
REAL32	00h	00h	C8h	40h
	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$

Figure A.4 - Transfer syntax of data type REAL32

A.2.4 Compound Data Types

A.2.4.1 General

Type definitions of compound data types expand to a unique list of type definitions involving only basic data types. Correspondingly, data of compound type *´type_name´* are ordered lists of component data named *´component_name_i´* of basic type *´basic_type_i´*.

Compound data types constructors are ARRAY and STRUCT OF.

```
STRUCT OF
            basic_type_1        component_name_1,
            basic_type_2        component_name_2,
            …                   …
            basic_type_N        component_name_N
type_name
```

```
ARRAY [ length ] OF basic_type          type_name
```

The bit sequence representing data of compound type is obtained by concatenating the bit sequences representing the component data.
Assume that the components *´component_name_i´* shall be represented by their bit sequences

b(i), for *i* = 1,…,N

Then the compound data is represented by the concatenated sequence

$b_0(1)$ .. $b_{n-1}(1)$ .. $b_{n-1}(N)$.

Example:
Consider the data type

STRUCT OF
INTEGER10 x,
UNSIGNED5 u
NewData

Assume x = - 423 = 259h and u = 30 = 1Eh. Let b(x) and b(u) denote the bit sequences representing the values of x and u, respectively. Then:

b(x)	= $b_0(x)$ .. $b_9(x)$	*= 1001101001*
b(u)	= $b_0(u)$ .. $b_4(u)$	*= 01111*
b(xu) = b(x) b(u)	= $b_0(xu)$ .. $b_{14}(xu)$	*= 1001101001 01111*

The value of the structure is transferred with two octets, first 59h and then 7Ah.

A.2.4.2 Extended Data Types

A.2.4.2.1 General

The extended data types consist of the basic data types and the compound data types defined in the following subsections.

A.2.4.2.2 Octet String

The data type OCTET_STRING*length* is defined below; *length* shall be the length of the octet string.

ARRAY [length] OF UNSIGNED8 OCTET_STRING*length*

A.2.4.2.3 Visible String

The data type VISIBLE_STRING*length* is defined below. The admissible values of data of type *VISIBLE_CHAR* shall be 0h and the range from 20h to 7Eh. The data shall be interpreted as ISO 646-1991(E) 7-bit coded characters. *length* shall be the length of the visible string.

UNSIGNED8	VISIBLE_CHAR
ARRAY [length] OF VISIBLE_CHAR	VISIBLE_STRING*length*

There is no 0h necessary to terminate the string.

A.2.4.2.4 Unicode String

The data type UNICODE_STRING*length* is defined below; *length* shall be the length of the unicode string.

ARRAY [length] OF UNSIGNED16 UNICODE_STRING*length*

A.2.4.2.5 Time of Day

The data type TIME_OF_DAY represents absolute time. It shall follow from the definition and the encoding rules that TIME_OF_DAY is represented as bit sequence of length 48.

Component ms shall be the time in milliseconds after midnight. Component days shall be the number of days since January 1, 1984.

```
STRUCT OF
        UNSIGNED28    ms,
        VOID4         reserved,
        UNSIGNED16    days
TIME_OF_DAY
```

A.2.4.2.6 Time Difference

The data type TIME_DIFFERENCE represents a time difference. It shall follow from the definition and the encoding rules that TIME_DIFFERENCE is represented as bit sequence of length 48.

Time differences are sums of numbers of days and milliseconds. Component ms shall be the number milliseconds. Component days shall be the number of days.

```
STRUCT OF
        UNSIGNED28    ms,
        VOID4         reserved,
        UNSIGNED16    days
TIME_DIFFERENCE
```

A.2.4.2.7 Domain

Domains may be used to transfer an arbitrary large block of data from a client to a server and vv. The contents of a data block is application specific and does not fall within the scope of this European Standard.

Annex B
(normative)

Object dictionary

B.1 General structure of the object dictionary

This section details the Object dictionary structure and entries which are common to all devices. The format of the Object dictionary entries used in this European Standard is shown in Table B.1.

Table B.1 - Format of Object dictionary Headings

Index (hex)	Object (Symbolic Name)	Name	Type	Attrib.	M/O

The complete Object dictionary consists of the six columns shown above. The Index column denotes the objects position within the Object dictionary. This acts as a kind of address to reference the desired data field. The sub-index is not specified here. The sub-index is used to reference data fields within a complex object such as an array or record.

The Object column contains the Object Name according to Table B.2 below and is used to denote what kind of object is at that particular index within the Object dictionary.

Table B.2 - Object dictionary Object Definitions

Object Name	Comments	Object code
NULL	A dictionary entry with no data fields	0
DOMAIN	Large variable amount of data e.g. executable program code	2
DEFTYPE	Denotes a type definition such as a Boolean, UNSIGNED16, float and so on	5
DEFSTRUCT	Defines a new record type e.g. the PDOMapping structure at 21h	6
VAR	A single value such as an UNSIGNED8, Boolean, float, Integer16, visible string etc.	7
ARRAY	A multiple data field object where each data field is a simple variable of the SAME basic data type e.g. array of UNSIGNED16 etc. Sub-index 0 is of UNSIGNED8 and therefore not part of the ARRAY data	8
RECORD	A multiple data field object where the data fields may be any combination of simple variables. Sub-index 0 is of UNSIGNED8 and therefore not part of the RECORD data	9

The name column provides a simple textual description of the function of that particular object. The type column gives information as to the type of the object. These include the following pre-defined types: BOOLEAN, floating point number, UNSIGNED Integer, Signed Integer, visible/octet string, time-of-day, time-difference and DOMAIN (see 0). It also includes the pre-defined complex data type PDOMapping and may also include others which are either manufacturer or device specific. It is not possible to define records of records, arrays of records or records with arrays as fields of that record. In the case where an object is an array or a record the sub-index is used to reference one data field within the object.

The Attribute column defines the access rights for a particular object (see Table B.3). The view point is from the bus into the device.

Table B.3 - Access attributes for data objects

Attribute	Description
rw	read and write access
wo	write only access
ro	read only access
Const	read only access, value is constant

Optional objects listed in the object dictionary with the attribute rw may be implemented as read only. Exceptions are defined in the detailed object specification.

The M/O column (see Table B.1) defines whether the object is mandatory or optional. A mandatory object shall be implemented on a device. An optional object may be implemented on a device. The support of certain objects or features however may require the implementation of related objects. In this case, the relations are described in the detailed object specification.

B.2 Dictionary components

The overall layout of the Object dictionary is shown in Table B.4.

Index 01h - 1Fh contain the standard data types, index 20h to 23h shall contain predefined complex data types. The range of indices from 24h to 3Fh shall not be defined yet but shall be reserved for future standard data structures.

The range of indices from 40h to 5Fh is free for manufacturers to define own complex data types. The range 60h to 7Fh shall contain device profile specific standard data types. From 80h to 9Fh device profile specific complex data types shall be defined. The range A0h to 25Fh shall be reserved for the data type definitions for Multiple Device Modules similar to the entries 60h to 9Fh. The entries form 360h to FFFh is reserved for future use. The range 1000h to 1FFFh shall contain the communication specific Object dictionary entries.

These parameters are called *communication entries,* their specification is common to all device types, regardless of the device profile they use. The objects in range 1000h to 1FFFh not specified by this European Standard is reserved for future use. The range 2000h to 5FFFh is free for manufacturer-specific profile definition.

The range 6000h to 9FFFh shall contain the standardised device profile parameters. The range A000h to FFFFh is reserved for future use.

B.3 Data Type Entry Specification

B.3.1 General

The static data types are placed in the Object dictionary for definition purposes only. However, indices in the range 0001h - 0007h may be mapped as well in order to define the appropriate space in the RPDO as not being used by this device (do not care). The indices 0009h to 000Bh, 000Fh shall not be mapped into a PDO.

The order of the data types is as shown in Table B.4.

Table B.4 - Object dictionary Data Types

Index	Object	Name
0001	DEFTYPE	BOOLEAN
0002	DEFTYPE	INTEGER8
0003	DEFTYPE	INTEGER16
0004	DEFTYPE	INTEGER32
0005	DEFTYPE	UNSIGNED8
0006	DEFTYPE	UNSIGNED16
0007	DEFTYPE	UNSIGNED32
0008	DEFTYPE	REAL32
0009	DEFTYPE	VISIBLE_STRING
000A	DEFTYPE	OCTET_STRING
000B	DEFTYPE	UNICODE_STRING
000C	DEFTYPE	TIME_OF_DAY
000D	DEFTYPE	TIME_DIFFERENCE
000E	DEFTYPE	reserved
000F	DEFTYPE	DOMAIN
0010	DEFTYPE	INTEGER24
0011	DEFTYPE	REAL64
0012	DEFTYPE	INTEGER40
0013	DEFTYPE	INTEGER48
0014	DEFTYPE	INTEGER56
0015	DEFTYPE	INTEGER64
0016	DEFTYPE	UNSIGNED24
0017	reserved	
0018	DEFTYPE	UNSIGNED40
0019	DEFTYPE	UNSIGNED48
001A	DEFTYPE	UNSIGNED56
001B	DEFTYPE	UNSIGNED64
001C-001F	reserved	
0020	DEFSTRUCT	PDO_COMMUNICATION_PARAMETER
0021	DEFSTRUCT	PDO_MAPPING
0022	DEFSTRUCT	SDO_PARAMETER
0023	DEFSTRUCT	IDENTITY
0024-003F	reserved	
0040-005F	DEFSTRUCT	Manufacturer Specific Complex Data Types
0060-007F	DEFTYPE	Device Profile (0) Specific Standard Data Types
0080-009F	DEFSTRUCT	Device Profile (0) Specific Complex Data Types
00A0-00BF	DEFTYPE	Device Profile 1 Specific Standard Data Types
00C0-00DF	DEFSTRUCT	Device Profile 1 Specific Complex Data Types
00E0-00FF	DEFTYPE	Device Profile 2 Specific Standard Data Types
0100-011F	DEFSTRUCT	Device Profile 2 Specific Complex Data Types
0120-013F	DEFTYPE	Device Profile 3 Specific Standard Data Types

Table B.4 - Object dictionary Data Types (continued)

Index	Object	Name
0140-015F	DEFSTRUCT	Device Profile 3 Specific Complex Data Types
0160-017F	DEFTYPE	Device Profile 4 Specific Standard Data Types
0180-019F	DEFSTRUCT	Device Profile 4 Specific Complex Data Types
01A0-01BF	DEFTYPE	Device Profile 5 Specific Standard Data Types
01C0-01DF	DEFSTRUCT	Device Profile 5 Specific Complex Data Types
01E0-01FF	DEFTYPE	Device Profile 6 Specific Standard Data Types
0200-021F	DEFSTRUCT	Device Profile 6 Specific Complex Data Types
0220-023F	DEFTYPE	Device Profile 7 Specific Standard Data Types
0240-025F	DEFSTRUCT	Device Profile 7 Specific Complex Data Types

The data type representations shall be as given in annex A. Every device does not need to support all the defined data types. A device only shall support the data types it uses with the objects in the range 1000h - 9FFFh.

The predefined complex data-types are placed after the standard data-types. Four types are defined at present, the *PDO CommPar* record *(PDO_COMMUNICATION_PARAMETER)*, the *PDO Mapping* record *(PDO_MAPPING)*, the *SDO Parameter* record *(SDO_PARAMETER)* and the *Identity* record *(IDENTITY)*. They are placed at index 20h, 21h, 22h and 23h.

For devices or device profiles that provide Multiple Device Modules like multiple axis controllers e.g. the DEFTYPE / DEFSTRUCT mechanism is enhanced for each virtual device with an offset of 40h for up to 7 additional virtual devices.

A device may provide the length of the standard data types encoded as UNSIGNED32 at read access to the index that refers to the data type. E.g. index 000Ch (Time of Day) contains the value 30h=48dec as the data type „Time of Day" is encoded using a bit sequence of 48 bit. If the length is variable (e.g. 000Fh = Domain), the entry contains 0h.

For the supported complex data types a device may provide the structure of that data type at read access to the corresponding data type index. Sub-index 0 then shall provide the number of entries at this index not counting sub-indices 0 and 255 and the following sub-indices shall contain the data type according to Table B.5 encoded as UNSIGNED8. The entry at Index 20h describing the structure of the PDO Communication Parameter then looks as shown in Table B.5 (see also objects 1400h to 15FFh):

Table B.5 - Complex data type example

Subindex	Value	(Description)
0h	04h	(4 sub indices follow)
1h	07h	(UNSIGNED32)
2h	05h	(UNSIGNED8)
3h	06h	(UNSIGNED16)
4h	05h	(UNSIGNED8)

Standard (simple) and complex manufacturer specific data types may be distinguished by attempting to read sub-index 1h: At a complex data type the device shall return a value and sub-index 0h shall contain the number of sub-indices that follow, at a standard data type the device aborts the SDO transfer as no sub-index 1h available.

Note that some entries of data type UNSIGNED32 have the character of a structure (e.g. PDO COB-ID entry, see Figure B.15).

B.3.2 Organisation of structured object dictionary entries

If an Object dictionary entry (index) contains several sub-indices, then sub-index 0 shall describe the highest available sub-index that follows, not considering FFh. This entry shall be encoded as UNSIGNED8.

Sub-index FFh describes the structure of the entry by providing the data type and the object type of the entry. It shall be encoded as UNSIGNED32 and organised as shown in Figure B.1.

	MSB		LSB
Bits	31-16	15-8	7-0
Value	Reserved (value: 00 00h)	Data Type (seeTable B.4)	Object (see Table B.2)
Encoded as	-	UNSIGNED8	UNSIGNED8

Figure B.1 - Structure of sub-index FFh

Sub-index FFh may be supported. If it is supported throughout the Object dictionary and the structure of the complex data types is provided as well, it enables the uploading of the entire structure of the Object dictionary.

B.4 Specification of predefined complex data types

B.4.1 General

This section describes the structure of the predefined complex data types used for communication. The value range and the meaning is explained at the detailed description of the objects using these types.

B.4.2 PDO communication parameter record specification

Table B.6 defines the PDO Communication Parameter Record.

Table B.6 - PDO Communication Parameter Record

Index	Sub-Index	Field in PDO Communication Parameter Record	Data Type
0020h	0h	number of supported entries in the record	UNSIGNED8
	1h	COB-ID	UNSIGNED32
	2h	transmission type	UNSIGNED8
	3h	inhibit time	UNSIGNED16
	4h	reserved	UNSIGNED8
	5h	event timer	UNSIGNED16

B.4.3 PDO mapping parameter record specification

Table B.7 defines the PDO Mapping Parameter Record.

Table B.7 - PDO Mapping Parameter Record

Index	Sub-Index	Field in PDO Parameter Mapping Record	Data Type
0021h	0h	number of mapped objects in PDO	UNSIGNED8
	1h	1st object to be mapped	UNSIGNED32
	2h	2nd object to be mapped	UNSIGNED32
: : : : :	: : : : : :	: : : : : : : :	: : : : : :
	40h	64th object to be mapped	UNSIGNED32

B.4.4 SDO parameter record specification

Table B.8 defines the SDO Parameter Record.

Table B.8 - SDO Parameter Record

Index	Sub-Index	Field in SDO Parameter Record	Data Type
0022h	0h	number of supported entries	UNSIGNED8
	1h	COB-ID client -> server	UNSIGNED32
	2h	COB-ID server -> client	UNSIGNED32
	3h	node ID of SDO’s client resp. server	UNSIGNED8

B.4.5 Identity record specification

Table B.9 defines the Identity Record.

Table B.9 - Identity Record

Index	Sub-Index	Field in Identity Record	Data Type
0023h	0h	number of supported entries	UNSIGNED8
	1h	Vendor-ID	UNSIGNED32
	2h	Product code	UNSIGNED32
	3h	Revision number	UNSIGNED32
	4h	Serial number	UNSIGNED32

B.5 Communication profile specification

B.5.1 Detailed object specification

The structure of the Object dictionary entries is described in the following manner: All device, interface and application profiles based on this communication profile shall follow this structure (see Table B.10 Table B.11

Table B.10 - Format of an Object Description

OBJECT DESCRIPTION

INDEX	Profile Index Number
Name	Name of parameter
Object Code	Variable classification
Data Type	Data type classification
Category	Optional or Mandatory

The Object Code shall be one of those defined in OBJECT DESCRIPTION Table B.10 above. For better readability, the Object Description additionally contains the symbolic Object Name.

Table B.11 - Object Value Description Format

ENTRY DESCRIPTION

Sub-Index	Number of sub-indices being described (field only used for Arrays, Records and Structures)
Description	Descriptive name of the Sub-Index (field only used for Arrays, Records and Structures)
Data Type	Data type classification (field only used for Records and Structures)
Entry Category	Specifies if the Entry is Optional or Mandatory or Conditional in case the Object is present
Access	Read Only (ro) or Read/Write (rw) or Write Only (wo) or Const. In OPERATIONAL state the Access to Object dictionary Entries may be limited, e.g set to ro.
PDO Mapping	Optional/Default/No - may this object be mapped to a PDO. Description: Optional: Object may be mapped into a PDO Default: Object is part of the default mapping (see device profile) No: Object shall not be mapped into a PDO
Value Range	range of possible values, or name of data type for full range
Default Value	No: not defined by this specification Value: default value of an object after device initialisation

For simple variables the value description shall appear once without the sub-index field and entry category. For complex data types the value description shall be defined for each element (sub-index).

B.5.2 Overview object dictionary entries for communication

Table B.12 gives an overview over the Object dictionary entries defined by the communication profile.

Table B.12 – Object Dictionary Entries of the Communication profile

Index (hex)	Object (Symbolic Name)	Name	Type	Acc.[a)]	M/O
1000	VAR	device type	UNSIGNED32	ro	M
1001	VAR	error register	UNSIGNED8	ro	M
1002	VAR	manufacturer status register	UNSIGNED32	ro	O
1003	ARRAY	pre-defined error field	UNSIGNED32	ro	O
1004	reserved for compatibility reasons				
1005	VAR	COB-ID SYNC	UNSIGNED32	rw	O
1006	VAR	communication cycle period	UNSIGNED32	rw	O
1007	VAR	synchronous window length	UNSIGNED32	rw	O
1008	VAR	manufacturer device name	Vis-String	const	O
1009	VAR	manufacturer hardware version	Vis-String	const	O
100A	VAR	manufacturer software version	Vis-String	const	O
100B	reserved for compatibility reasons				
100C	VAR	guard time	UNSIGNED16	rw	O
100D	VAR	life time factor	UNSIGNED8	rw	O
100E	reserved for compatibility reasons				
100F	reserved for compatibility reasons				
1010	ARRAY	store parameters	UNSIGNED32	rw	O
1011	ARRAY	restore default parameters	UNSIGNED32	rw	O
1012	VAR	COB-ID TIME	UNSIGNED32	rw	O
1013	VAR	high resolution time stamp	UNSIGNED32	rw	O
1014	VAR	COB-ID EMCY	UNSIGNED32	rw	O
1015	VAR	Inhibit Time EMCY	UNSIGNED16	rw	O
1016	ARRAY	Consumer heartbeat time	UNSIGNED32	rw	O
1017	VAR	Producer heartbeat time	UNSIGNED16	rw	O
1018	RECORD	Identity Object	Identity (23h)	ro	M
1019		reserved			
:::::	:::::	:::::	:::::	:::::	:::::
11FF		reserved			
Server SDO Parameter					
1200	RECORD	1st Server SDO parameter	SDO Parameter (22h)	ro	O
1201	RECORD	2nd Server SDO parameter	SDO Parameter (22h)	rw	M/O**
:::::	:::::	:::::	:::::	:::::	:::::
127F	RECORD	128th Server SDO parameter	SDO Parameter (22h)	rw	M/O**
Client SDO Parameter					
1280	RECORD	1st Client SDO parameter	SDO Parameter (22h)	rw	M/O**
1281	RECORD	2nd Client SDO parameter	SDO Parameter (22h)	rw	M/O**
:::::	:::::	:::::	:::::	:::::	:::::
12FF	RECORD	128th Client SDO parameter	SDO Parameter (22h)	rw	M/O**
1300		reserved			
:::::	:::::	:::::	:::::	:::::	:::::
13FF		reserved			

Table B.12 – Object Dictionary Entries of the Communication profile (continued)

Index (hex)	**Object (Symbolic Name)**	**Name**	**Type**	**Acc.[a)]**	**M/O**
		Receive PDO Communication Parameter			
1400	RECORD	1st receive PDO Parameter	PDO CommPar (20h)	rw	M/O*
1401	RECORD	2nd receive PDO Parameter	PDO CommPar (20h)		M/O*
:::::	:::::	:::::	:::::	:::::	:::::
15FF	RECORD	512th receive PDO Parameter	PDO CommPar (20h)	rw	M/O*
		Receive PDO Mapping Parameter			
1600	RECORD	1st receive PDO mapping	PDO Mapping (21h)	rw	M/O*
1601	RECORD	2nd receive PDO mapping	PDO Mapping (21h)	rw	M/O*
:::::	:::::	:::::	:::::	:::::	:::::
17FF	RECORD	512th receive PDO mapping	PDO Mapping (21h)	rw	M/O*
		Transmit PDO Communication Parameter			
1800	RECORD	1st transmit PDO Parameter	PDO CommPar (20h)	rw	M/O*
1801	RECORD	2nd transmit PDO Parameter	PDO CommPar (20h)	rw	M/O*
:::::	:::::	:::::	:::::	:::::	::::
19FF	RECORD	512th transmit PDO Parameter	PDO CommPar (20h)	rw	M/O*
		Transmit PDO Mapping Parameter			
1A00	RECORD	1st transmit PDO mapping	PDO Mapping (21h)	rw	M/O*
1A01	RECORD	2nd transmit PDO mapping	PDO Mapping (21h)	rw	M/O*
:::::	:::::	:::::	:::::	:::::	:::::
1BFF	RECORD	512th transmit PDO mapping	PDO Mapping (21h)	rw	M/O*

* If a device supports PDOs, the according PDO communication parameter and PDO mapping entries in the Object dictionary shall be implemented. These may be read_only.

** If a device supports SDOs, the according SDO parameters in the Object dictionary shall be implemented.

a) Access type listed here may vary for certain sub-indices. See detailed object specification.

B.5.3 Detailed specification of communication profile specific objects

Object 1000h: Device Type

Shall contain information about the device type. The object at index 1000h describes the type of device and its functionality. It shall be composed of a 16-bit field which describes the device profile that is used and a second 16-bit field which gives additional information about optional functionality of the device (see Figure B.2). The Additional Information parameter is device profile specific. Its specification does not fall within the scope of this European Standard, it is defined in the appropriate device profile. The value 0x0000 shall indicate a device that does not follow a standardised device profile.

For multiple device modules the Additional Information parameter shall contain FFFFh and the device profile number referenced by object 1000h is the device profile of the first device in the Object dictionary. All other devices of a multiple device module shall identify their profiles at objects 67FFh + x * 800h with x = internal number of the device (0 – 7) . These entries describe the device type of the preceding device.

OBJECT DESCRIPTION

INDEX	1000h
Name	device type
Object Code	VAR
Data Type	UNSIGNED32
Category	Mandatory

ENTRY DESCRIPTION

Access	ro
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	No

Byte:	MSB	LSB
	Additional Information	Device Profile Number

Figure B.2 - Structure of the Device Type Parameter

Object 1001h: Error Register

This object is an error register for the device. The device may map internal errors in this byte. This entry shall be implemented for all devices. It is a part of an Emergency object.

OBJECT DESCRIPTION

INDEX	1001h
Name	error register
Object Code	VAR
Data Type	UNSIGNED8
Category	Mandatory

ENTRY DESCRIPTION

Access	ro
PDO Mapping	Optional
Value Range	UNSIGNED8
Default Value	No

Table B.13 - Structure of the Error Register

Bit	M/O	Meaning
0	M	generic error
1	O	current
2	O	voltage
3	O	temperature
4	O	communication error (overrun, error state)
5	O	device profile specific
6	O	Reserved (always 0)
7	O	manufacturer specific

If a bit is set to 1 the specified error has occurred. The only error that shall be signalled is the generic error. The generic error shall be signalled at any error situation.

Object 1002h: Manufacturer Status Register

This object is a common status register for manufacturer specific purposes. In this European Standard only the size and the location of this object is defined.

OBJECT DESCRIPTION

INDEX	1002h
Name	manufacturer status register
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION

Access	ro
PDO Mapping	Optional
Value Range	UNSIGNED32
Default Value	No

Object 1003h: Pre-defined Error Field

The object at index 1003h shall hold the errors that have occurred on the device and have been signalled via the Emergency Object. In doing so it provides an error history.

1. The entry at sub-index 0 shall contain the number of actual errors that are recorded in the array starting at sub-index 1.
2. Every new error shall be stored at sub-index 1, the older ones shall move down the list.
3. Writing a „0“ to sub-index 0 deletes the entire error history (empties the array). Values higher than 0 are not allowed to write. This shall lead to an abort message (error code: 0609 0030h)
4. The error numbers shall be of type UNSIGNED32 (see Table B.13) and shall be composed of a 16 bit error code and a 16 bit additional error information field which is manufacturer specific (see Figure B.3). The error code shall be contained in the lower 2 bytes (LSB) and the additional information shall be included in the upper 2 bytes (MSB). If this object is supported it shall consist of two entries at least: The length entry on sub-index 0h and at least one error entry at sub-index 1H.

Byte: MSB LSB

Additional Information	Error code

Figure B.3 - Structure of the pre-defined error field

OBJECT DESCRIPTION

INDEX	1003h
Name	pre-defined error field
Object Code	ARRAY
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION

Sub-Index	0h
Description	number of errors
Entry Category	Mandatory
Access	rw
PDO Mapping	No
Value Range	0 - 254
Default Value	0

Sub-Index	1h
Description	standard error field
Entry Category	Optional
Access	ro
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	No

Sub-Index	2h - FEh
Description	standard error field
Entry Category	Optional
Access	ro
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	No

Object 1005h: COB-ID SYNC message

Index 1005h shall define the COB-ID of the Synchronisation Object (SYNC). Further, it defines whether the device generates the SYNC. The structure of this object is shown in Figure B.4 and Table B.14.

UNSIGNED32

MSB LSB

bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	X	0/1	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	X	0/1	1	29 -bit Identifier	

Figure B.4 - Structure of SYNC COB-ID entry

Table B.14 - Description of SYNC COB-ID entry

bit number	value	meaning
31 (MSB)	X	do not care
30	0 1	Device does not generate SYNC message Device generates SYNC message
29	0 1	11-bit ID (CAN 2.0A) 29-bit ID (CAN 2.0B)

28 – 11	0
	X
		if bit 29=0
		if bit 29=1: bits 28-11 of 29-bit-SYNC-COB-ID

10-0 (LSB)	X	bits 10-0 of SYNC-COB-ID

Bits 29, 30 may be static (not changeable). If a device is not able to generate SYNC messages, an attempt to set bit 30 shall be responded with an abort message (abort code: 0609 0030h). Devices supporting the standard CAN frame type only either ignore attempts to change bit 29 or respond with an abort message (abort code: 0609 0030h). The first transmission of SYNC object starts within 1 sync cycle after setting Bit 30 to 1. Bit 0-29 shall not be changed, while the objects exist (Bit 30=1).

OBJECT DESCRIPTION

INDEX	1005h
Name	COB-ID SYNC
Object Code	VAR
Data Type	UNSIGNED32
Category	Conditional; Mandatory, if PDO communication on a synchronous base is supported

ENTRY DESCRIPTION

Access	rw
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	80h or 8000 0080h

Object 1006h: Communication Cycle Period

This object shall define the communication cycle period in µs. This period defines the SYNC interval. It shall be 0 if not used. If the communication cycle period on sync producer is changed to a new value unequal 0 the transmission of sync object shall resume within 1 sync cycle of the new value.

OBJECT DESCRIPTION

INDEX	1006h
Name	communication cycle period
Object Code	VAR
Data Type	UNSIGNED32
Category	Conditional; Mandatory for Sync producers

ENTRY DESCRIPTION

Access	rw
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	0

Object 1007h: Synchronous Window Length

Shall contain the length of the time window for synchronous PDOs in µs. It shall be 0 if not used.

OBJECT DESCRIPTION

INDEX	1007h
Name	synchronous window length
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION

Access	rw
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	0

Object 1008h: Manufacturer Device Name

Shall contain the manufacturer device name.

OBJECT DESCRIPTION

INDEX	1008h
Name	manufacturer device name
Object Code	VAR
Data Type	Visible String
Category	Optional

ENTRY DESCRIPTION

Access	const
PDO Mapping	No
Value Range	No
Default Value	No

Object 1009h: Manufacturer Hardware Version

Shall contain the manufacturer hardware version description.

OBJECT DESCRIPTION

INDEX	1009h
Name	manufacturer hardware version
Object Code	VAR
Data Type	Visible String
Category	Optional

ENTRY DESCRIPTION

Access	const
PDO Mapping	No
Value Range	No
Default Value	No

Object 100Ah: Manufacturer Software Version

Shall contain the manufacturer software version description.

OBJECT DESCRIPTION

INDEX	100Ah
Name	manufacturer software version
Object Code	VAR
Data Type	Visible String
Category	Optional

ENTRY DESCRIPTION

Access	const
PDO Mapping	No
Value Range	No
Default Value	No

Object 100Ch: Guard Time

The objects at index 100Ch and 100Dh shall include the guard time in milliseconds and the life time factor. The life time factor multiplied with the guard time gives the life time for the Life Guarding Protocol. It shall be 0 if not used.

OBJECT DESCRIPTION

INDEX	100Ch
Name	guard time
Object Code	VAR
Data Type	UNSIGNED16
Category	Conditional; Mandatory, if heartbeat is not supported

ENTRY DESCRIPTION

Access	rw; ro, if life guarding is not supported
PDO Mapping	No
Value Range	UNSIGNED16
Default Value	0

Object 100Dh: Life Time Factor

The life time factor multiplied with the guard time gives the life time for the node guarding protocol. It shall be 0 if not used.

OBJECT DESCRIPTION

INDEX	100Dh
Name	life time factor
Object Code	VAR
Data Type	UNSIGNED8
Category	Conditional; Mandatory, if heartbeat is not supported

ENTRY DESCRIPTION

Access	rw; ro, if life guarding is not supported
PDO Mapping	No
Value Range	UNSIGNED8
Default Value	0

Object 1010h: Store parameters

This object supports the saving of parameters in non volatile memory. By read access the device provides information about its saving capabilities. Several parameter groups are distinguished:

Sub-index 0 shall contain the largest Sub-Index that is supported.

Sub-index 1 shall refer to all parameters that may be stored on the device.

Sub-index 2 shall refer to communication related parameters (Index 1000h - 1FFFh manufacturer specific communication parameters).

Sub-index 3 shall refer to application related parameters (Index 6000h - 9FFFh manufacturer specific application parameters).

At Sub-index 4 - 127 manufacturers may store their choice of parameters individually.

Sub-index 128 - 254 shall be reserved for future use.

In order to avoid storage of parameters by mistake, storage shall be only executed when a specific signature is written to the appropriate Sub-Index. The signature shall be „save“ (see Figure B.5).

Signature	MSB			LSB
ISO 8859 (“ASCII”)	e	v	a	s
hex	65h	76h	61h	73h

Figure B.5 - Storage write access signature

On reception of the correct signature in the appropriate sub-index the device shall store the parameter and then shall confirm the SDO transmission (initiate download response). If the storing failed, the device shall respond with an Abort SDO Transfer (abort code: 0606 0000h).

If a wrong signature is written, the device refuses to store and responds with Abort SDO Transfer (abort code: 0800 002xh).

On read access to the appropriate Sub-Index the device shall provide information about its storage functionality with the format shown in Figure B.6 and Table B.15

	UNSIGNED32 MSB	LSB	
bits	*31-2*	*1*	*0*
	reserved (=0)	0/1	0/1

Figure B.6 - Storage read access structure

Table B.15 - Structure of read access

bit number	value	meaning
31-2	0	*reserved (=0)*
1	0 1	Device does not save parameters autonomously Device saves parameters autonomously
0	0 1	Device does not save parameters on command Device saves parameters on command

Autonomous saving means that a device stores the storable parameters in a non-volatile manner without user request.

OBJECT DESCRIPTION

INDEX	1010h
Name	store parameters
Object Code	ARRAY
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION

Sub-Index	0h
Description	largest subindex supported
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	1h – 7Fh
Default Value	No

Sub-Index	1h
Description	save all parameters
Entry Category	Mandatory
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.5 for write access; Figure B.6 for read access)
Default Value	No

Sub-Index	2h
Description	save communication parameters
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.5 for write access; Figure B.6 for read access)
Default Value	No

Sub-Index	3h
Description	save application parameters
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.5 for write access; Figure B.6 for read access)
Default Value	No

Sub-Index	4h - 7Fh
Description	save manufacturer defined parameters
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.5 for write access; Figure B.6 for read access)
Default Value	No

Object 1011h: Restore default parameters

With this object the default values of parameters according to the communication or device profile are restored. By read access the device provides information about its capabilities to restore these values. Several parameter groups are distinguished.

Sub-index 0 shall contain the largest Sub-Index that is supported.

Sub-index 1 shall refer to all parameters that may be restored.

Sub-index 2 shall refer to communication related parameters (Index 1000h - 1FFFh manufacturer specific communication parameters).

Sub-index 3 shall refer to application related parameters (Index 6000h - 9FFFh manufacturer specific application parameters).

At sub-index 4 - 127 manufacturers may restore their individual choice of parameters.

Sub-index 128 - 254 shall be reserved for future use.

In order to avoid the restoring of default parameters by mistake, restoring shall be only executed when a specific signature is written to the appropriate sub-index. The signature shall be „load“ (see Figure B.7).

Signature	MSB			LSB
ASCII	d	a	o	l
hex	64h	61h	6Fh	6Ch

Figure B.7 - Restoring write access signature

On reception of the correct signature in the appropriate sub-index the device restores the default parameters and then confirms the SDO transmission (initiate download response). If the restoring failed, the device responds with an Abort SDO Transfer (abort code: 0606 0000h). If a wrong signature is written, the device shall refuse to restore the defaults and shall respond with an Abort SDO Transfer (abort code: 0800 002xh).

The default values are set valid after the device is reset (reset node for sub-index 1h – 7Fh, reset communication for sub-index 2h) or power cycled.

Figure B.8 shows the 'restore' procedure.

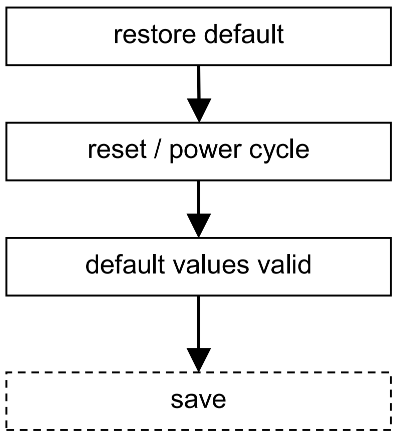


Figure B.8 - Restore procedure

On read access to the appropriate sub-index the device provides information about its default parameter restoring capability with the format shown in Figure B.9 and Table B.16

UNSIGNED32

	MSB LSB	
bits	*31-1*	*0*
	reserved (=0)	0/1

Figure B.9 - Restoring default values read access structure

Table B.16 - Structure of restore read access

bit number	value	meaning
31-1	0	*reserved (=0)*
0	0 1	Device does not restore default parameters Device restores parameters

OBJECT DESCRIPTION

INDEX	1011h
Name	restore default parameters
Object Code	ARRAY
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION

Sub-Index	0h
Description	largest subindex supported
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	1h – 7Fh
Default Value	No

Sub-Index	1h
Description	restore all default parameters
Entry Category	Mandatory
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.7)
Default Value	No

Sub-Index	2h
Description	restore communication default parameters
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.7)
Default Value	No

Sub-Index	3h
Description	restore application default parameters
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.7)
Default Value	No

Sub-Index	4h - 7Fh
Description	restore manufacturer defined default parameters
Entry Category	Optional
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.7)

Object 1012h: COB-ID Time Stamp Object

Index 1012h shall define the COB-ID of the Time-Stamp Object (TIME). Further, it defines whether the device consumes the TIME or whether the device generates the TIME. The structure of this object is shown in Figure B.10 and Table B.17.

UNSIGNED32

	MSB				LSB
bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	0/1	0/1	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	0/1	0/1	1	29 -bit Identifier	

Figure B.10 - Structure of TIME COB-ID entry

Table B.17 - Description of TIME COB-ID entry

bit number	value	Meaning
31 (MSB)	0	Device does not consume TIME message
	1	Device consumes TIME message
30	0	Device does not produce TIME message
	1	Device produces TIME message
29	0	11-bit ID (CAN 2.0A)
	1	29-bit ID (CAN 2.0B)
28 – 11	0	if bit 29=0
	X	if bit 29=1: bits 28-11 of 29-bit-TIME-COB-ID
10-0 (LSB)	X	bits 10-0 of TIME-COB-ID

Bits 29, 30 may be static (not changeable). If a device is not able to generate TIME messages, an attempt to set bit 30 is responded with an abort message (abort code: 0609 0030h). For devices supporting the standard CAN frame type only, an attempt to set bit 29 shall be responded with an abort message (abort code: 0609 0030h). Bit 0-29 shall not be changed, while the objects exist (Bit 30=1).

OBJECT DESCRIPTION

INDEX	1012h
Name	COB-ID time stamp message
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION

Access	rw
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	100h

Object 1013h: High Resolution Time Stamp

This object shall contain a time stamp with a resolution of 1 μs (see 5.2.2). It may be mapped into a PDO in order to define a high resolution time stamp message. (Note that the data type of the standard time stamp message (TIME) is fixed). Further application specific use is encouraged.

OBJECT DESCRIPTION

INDEX	1013h
Name	high resolution time stamp
Object Code	VAR
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION

Access	rw
PDO Mapping	Optional
Value Range	UNSIGNED32
Default Value	0

Object 1014h: COB-ID Emergency Object

Index 1014h shall define the COB-ID of the Emergency Object (EMCY). The structure of this object is shown in Figure B.11 and the description is given in Table B.18.

UNSIGNED32

	MSB				LSB
bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	0/1	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	0/1	0	1	29 -bit Identifier	

Figure B.11 - Structure of the EMCY Identifier entry

Table B.18 - Description of EMCY COB-ID entry

bit number	value	Meaning
31 (MSB)	0 1	EMCY exists / is valid EMCY does not exist / is not valid
30	0	reserved (always 0)
29	0 1	11-bit ID (CAN 2.0A) 29-bit ID (CAN 2.0B)

28 - 11	0
	X
		if bit 29=0
		if bit 29=1: bits 28-11 of 29-bit-COB-ID

10-0 (LSB)	X	bits 10-0 of COB-ID

For Devices supporting the standard CAN frame type only, an attempt to set bit 29 shall be responded with an abort message (abort code: 0609 0030h). Bits 0-29 shall not be changed, while the object exists (Bit 31=0).

OBJECT DESCRIPTION

INDEX	1014h
Name	COB-ID Emergency message
Object Code	VAR
Data Type	UNSIGNED32
Category	Conditional; Mandatory, if Emergency is supported

ENTRY DESCRIPTION

Access	ro; optinal rw
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	80h + Node-ID

Object 1015h: Inhibit Time EMCY

The inhibit time for the EMCY message may be adjusted via this entry. If this entry exists it shall be writable in the object dictionary. The time shall be a multiple of 100µs.

OBJECT DESCRIPTION

INDEX	1015h
Name	Inhibit Time EMCY
Object Code	VAR
Data Type	UNSIGNED16
Category	Optional

ENTRY DESCRIPTION

Access	rw
PDO Mapping	No
Value Range	UNSIGNED16
Default Value	0

Object 1016h: Consumer Heartbeat Time

The consumer heartbeat time shall define the expected heartbeat cycle time and thus shall be higher than the corresponding producer heartbeat time configured on the device producing this heartbeat (see Figure B.12). Monitoring starts after the reception of the first heartbeat. If the consumer heartbeat time shall be 0 the corresponding entry is not used. The time shall be a multiple of 1ms.

UNSIGNED32

	MSB		LSB
Bits	31-24	23-16	15-0
Value	reserved (value: 00h)	Node-ID	heartbeat time
Encoded as	-	UNSIGNED8	UNSIGNED16

Figure B.12 - Structure of Consumer Heartbeat Time entry

At an attempt to configure several consumer heartbeat times unequal 0 for the same Node-ID the device aborts the SDO download with abort code 0604 0043h

OBJECT DESCRIPTION

INDEX	1016h
Name	Consumer Heartbeat Time
Object Code	ARRAY
Data Type	UNSIGNED32
Category	Optional

ENTRY DESCRIPTION

Sub-Index	0h
Description	number entries
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	1 – 127
Default Value	No

Sub-Index	1h
Description	Consumer Heartbeat Time
Entry Category	Mandatory
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.12)
Default Value	0

Sub-Index	2h – 7Fh
Description	Consumer Heartbeat Time
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.12)
Default Value	No

Object 1017h: Producer Heartbeat Time

The producer heartbeat time shall define the cycle time of the heartbeat. The producer heartbeat time shall be 0 if it not used. The time shall be a multiple of 1ms.

OBJECT DESCRIPTION

INDEX	1017h
Name	Producer Heartbeat Time
Object Code	VAR
Data Type	UNSIGNED16
Category	Conditional; Mandatory if guarding not supported

ENTRY DESCRIPTION

Access	rw
PDO Mapping	No
Value Range	UNSIGNED16
Default Value	0

Object 1018h: Identity Object

The object at index 1018h shall contain general information about the device.

The Vendor ID (sub-index 1h) shall contain a unique value.

The manufacturer-specific Product code (sub-index 2h) shall identify a specific device version.

The manufacturer-specific Revision number (sub-index 3h) shall consist of a major revision number and a minor revision number (see Figure B.13). The major revision number identifies a specific CANopen behaviour. If the CANopen functionality is expanded, the major revision shall be incremented. The minor revision number identifies different versions with the same CANopen behaviour.

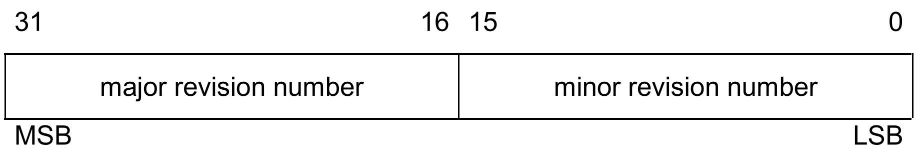


Figure B.13 - Structure of Revision number

The manufacturer-specific Serial number (sub-index 4h) identifies a specific device.

OBJECT DESCRIPTION

INDEX	1018h
Name	Identity Object
Object Code	RECORD
Data Type	Identity
Category	Mandatory

ENTRY DESCRIPTION

Sub-Index	0h
Description	number of entries
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	1 .. 4
Default Value	No

Sub-Index	1h
Description	Vendor ID
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	No

Sub-Index	2h
Description	Product code
Entry Category	Optional
Access	ro
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	No

Sub-Index	3h
Description	Revision number
Entry Category	Optional
Access	ro
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	No

Sub-Index	4h
Description	Serial number
Entry Category	Optional
Access	ro
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	No

Object 1200h - 127Fh: Server SDO Parameter

In order to describe the SDOs used on a device the data type SDO Parameter is introduced. The data type shall have the index 22h in the Object dictionary. The structure shall be as described in annex A.

The number of supported entries in the SDO object record shall be specified at sub-index 0h. The values at 1h and 2h specify the COB-ID for this SDO (see Figure B.14). Sub-index 3 shall give the server of the SDO in case the record describes an SDO for which the device is client and gives the client of the SDO if the record describes an SDO for which the device is server.

UNSIGNED32

	MSB				LSB
bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	0/1	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	0/1	0	1	29-bit Identifier	

Figure B.14 - Structure of SDO COB-ID entry

Table B.19 - Description of SDO COB-ID entry

bit number	Value	Meaning
31 (MSB)	0 1	SDO exists / is valid SDO does not exist/ is not valid
30	0	reserved (always 0)
29	0 1	11-bit ID (CAN 2.0A) 29-bit ID (CAN 2.0B)

28 - 11	0
	X
		if bit 29=0
		if bit 29=1: bits 28-11 of 29-bit-COB-ID

10-0 (LSB)	X	bits 10-0 of COB-ID

An SDO shall be only valid if both SDO-valid-bits are 0. For devices supporting the standard CAN frame type only, an attempt to set bit 29 shall be responded with an abort message (abort code: 0609 0030h).

These objects shall contain the parameters for the SDOs for which the device is the server. If a device handles more than one server SDO the default SDO shall be located at index 1200h as the first server SDO. This entry is read only. It shall be ensured that the COB-IDs of the default SDO shall not be manipulated by writing to the Object dictionary. All additional server SDOs are invalid by default (invalid bit – see Table B.19), there description is located at subsequent indexes. The COB-ID shall not be changed, while the SDO exists.

The description of the Client of the SDO (sub-index 3h) may be implemented. It shall be not available for the default SDO (no Sub-index 3h at Index 1200h), as this entry is read only.

OBJECT DESCRIPTION

INDEX	1200h - 127Fh
Name	Server SDO parameter
Object Code	RECORD
Data Type	SDO Parameter
Category	Conditional Index 1200h: Optional Index 1201h - 127Fh: Mandatory for each additionally supported server SDO

ENTRY DESCRIPTION

Sub-Index	0h
Description	number of entries
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	Index 1200h: 2 Index 1201h – 127F: 2 - 3
Default Value	No

Sub-Index	1h
Description	COB-ID Client->Server (rx)
Entry Category	Mandatory
Access	Index 1200h: ro, Index 1201h-127Fh: rw
PDO Mapping	No
Value Range	UNSIGNED32 (Table B.19)
Default Value	Index 1200h: 600h+Node-ID, Index 1201h-127Fh: No

Sub-Index	2h
Description	COB-ID Server -> Client (tx)
Entry Category	Mandatory
Access	Index 1200h: ro Index 1201-127Fh: rw
PDO Mapping	No
Value Range	UNSIGNED32 (Table B.19)
Default Value	Index 1200h: 580h+Node-ID, Index 1201h-127Fh: No

Sub-Index	3h
Description	node ID of the SDO client
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	1h – 7Fh
Default Value	No

Object 1280h - 12FFh: Client SDO Parameter

These objects shall contain the parameters for the SDOs for which the device is the client. If the entry is supported, all sub-indices shall be available. Starting at index 1280h and subsequent indices. The entries are described at the Server SDO Parameter.

OBJECT DESCRIPTION

INDEX	1280h - 12FFh
Name	Client SDO parameter
Object Code	RECORD
Data Type	SDO Parameter
Category	Conditional; Mandatory for each supported client SDO

ENTRY DESCRIPTION

Sub-Index	0h
Description	number of entries
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	3
Default Value	3

Sub-Index	1h
Description	COB-ID Client->Server (tx)
Entry Category	Mandatory
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Table B.19)
Default Value	No

Sub-Index	2h
Description	COB-ID Server -> Client (rx)
Entry Category	Mandatory
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32 (Table B.19)
Default Value	No

Sub-Index	3h
Description	node ID of the SDO server
Entry Category	Mandatory
Access	rw
PDO Mapping	No
Value Range	1h – 7Fh
Default Value	No

Object 1400h - 15FFh: Receive PDO Communication Parameter

Shall contain the communication parameters for the PDOs the device is able to receive. The type of the PDO communication parameter (20h) shall be as described in annex A. The sub-index 0h shall contain the number of valid entries within the communication record. Its value shall be at least 2. If inhibit time supported the value shall be3. At sub-index 1h resides the COB-ID of the PDO. This entry has been defined as UNSIGNED32 in order to cater for 11-bit CAN Identifiers (CAN 2.0A) as well as for 29-bit CAN identifiers (CAN 2.0B). The entry shall be interpreted as defined in Figure B.15 and Table B.20.

UNSIGNED32

	MSB				LSB
bits	*31*	*30*	*29*	*28-11*	*10-0*
11-bit-ID	0/1	0/1	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11-bit Identifier
29-bit-ID	0/1	0/1	1	29-bit Identifier	

Figure B.15 - Structure of PDO COB-ID entry

Table B.20 - Description of PDO COB-ID entry

bit number	Value	meaning
31 (MSB)	0	PDO exists / is valid
	1	PDO does not exist / is not valid
30	0	RTR allowed on this PDO
	1	no RTR allowed on this PDO
29	0	11-bit ID (CAN 2.0A)
	1	29-bit ID (CAN 2.0B)
28 – 11	0	if bit 29=0
	X	if bit 29=1: bits 28-11 of 29-bit-COB-ID
10-0 (LSB)	X	bits 10-0 of COB-ID

The PDO valid/not valid allows to select which PDOs are used in the operational state. There may be PDOs fully configured (e.g. by default) but not used, and therefore set to "not valid" (deleted). The feature is necessary for devices supporting more than 4 RPDOs or 4 TPDOs, because each device has only default identifiers for the first four RPDOs/TPDOs. Devices supporting the standard CAN frame type only or do not support Remote Frames, an attempt to set bit 29 to 1 or bit 30 to 0 is responded with an abort message (abort code: 0609 0030h). Bits 0-29 shall not be changed, while the PDO exists (Bit 31=0).

The transmission type (sub-index 2) defines the transmission/reception character of the PDO (see 5.1.2.2). Table B.21 describes the usage of this entry. On an attempt to change the value of the transmission type to a value that is not supported by the device an abort message (abort code: 0609 0030h) shall be generated.

Table B.21 - Description of transmission type

transmission type	PDO transmission				
	cyclic	acyclic	synchronous	asynchronous	RTR only
0		X	X		
1-240	X		X		
241-251	- reserved -				
252			X		X
253				X	X
254				X	
255				X	

Synchronous (transmission types 0-240 and 252) means that the transmission of the PDO shall be related to the SYNC object as described in 5.2. Preferably the devices use the SYNC as a trigger to output or actuate based on the previous synchronous Receive PDO, respectively to update the data transmitted at the following synchronous Transmit PDO. Details of this mechanism depend on the device type and are defined in the device profile if applicable.

Asynchronous means that the transmission of the PDO is not related to the SYNC object.

A transmission type of zero means that the message shall be transmitted synchronously with the SYNC object but not periodically.

A value between 1 and 240 means that the PDO is transferred synchronously and cyclically, the transmission type indicating the number of SYNC which are necessary to trigger PDO transmissions/receptions.

The transmission types 252 and 253 mean that the PDO is only transmitted on remote transmission request. At transmission type 252, the data is updated (but not sent) immediately after reception of the SYNC object. At transmission type 253 the data is updated at the reception of the remote transmission request (hardware and software restrictions may apply). These values are only possible for TPDOs.

For TPDOs transmission type 254 means, the application event is manufacturer specific (manufacturer specific part of the Object dictionary), transmission type 255 means, the application event is defined in the device profile. RPDOs with that type trigger the update of the mapped data with the reception.

Sub-index 3h shall contain the inhibit time. This time is a minimum interval for PDO transmission. The value is defined as multiple of 100µs. It is not allowed to change the value, while the PDO exists (Bit 31 of sub-index 1h is 0).

Sub-index 4h is reserved. If not implemented, read or write access shall lead to Abort SDO Transfer (abort code: 0609 0011h).

In mode 254/255 additionally an event time may be used for TPDO. If an event timer exists for a TPDO (value not equal to 0) the elapsing of the time shall be considered to be an event. The event timer shall elapse as multiple of 1 ms of the entry in sub-index 5h of the TPDO. This event shall cause the transmission of this TPDO in addition to otherwise defined events. The occurrence of the events shall set the timer.

Independent of the transmission type the RPDO event timer is used recognise the expiration of the RPDO.

OBJECT DESCRIPTION

INDEX	1400h - 15FFh
Name	receive PDO parameter
Object Code	RECORD
Data Type	PDO CommPar
Category	Conditional; Mandatory for each supported PDO

ENTRY DESCRIPTION

Sub-Index	0h
Description	largest sub-index supported
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	2 – 5

Sub-Index	1h
Description	COB-ID used by PDO
Entry Category	Mandatory
Access	ro; rw if variable COB-ID is supported
PDO Mapping	No
Value Range	UNSIGNED32 (Table B.20)
Default Value	Index 1400h: 200h + Node-ID, Index 1401h: 300h + Node-ID, Index 1402h: 400h + Node-ID, Index 1403h: 500h + Node-ID, Index 1404h – 15FFh: disabled

Sub-Index	2h
Description	transmission type
Entry Category	Mandatory
Access	ro; rw if variable transmission type is supported
PDO Mapping	No
Value Range	UNSIGNED8 (Table B.21)
Default Value	(Device Profile dependent)

Sub-Index	3h
Description	inhibit time (not used for RPDO)
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED16
Default Value	No

Sub-Index	4h
Description	compatibility entry
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED8
Default Value	No

Sub-Index	5h
Description	event timer
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	0 – not used UNSIGNED16
Default Value	No

Object 1600h - 17FFh: Receive PDO Mapping Parameter

Shall contain the mapping for the PDOs the device is able to receive. The type of the PDO mapping parameter (21h) shall be as described in annex A. The sub-index 0h contains the number of valid entries within the mapping record. This number of entries is also the number of the application variables which shall be transmitted/received with the corresponding PDO. The sub-indices from 1h to number of entries contain the information about the mapped application variables. These entries describe the PDO contents by their index, sub-index and length (Figure B.16). All three values shall be hexadecimal coded. The length entry shall contain the length of the object in bit (1..40h). This parameter may be used to verify the overall mapping length. It shall be implemented.

The structure of the entries from sub-index 1h – 40h shall be as shown in Figure B.16.

Byte:	MSB		LSB
	index (16 bit)	sub-index (8 bit)	object length (8 bit)

Figure B.16 - Structure of PDO Mapping Entry

If the change of the PDO mapping cannot be executed (e.g. the PDO length is exceeded or the SDO client attempts to map an object that cannot be mapped) the device shall respond with an Abort SDO Transfer Service.

Sub-index 0 determines the valid number of objects that have been mapped. For changing the PDO mapping first the PDO shall be deleted, the sub-index 0 shall be set to 0 (mapping is deactivated).

Then the objects may be remapped. When a new object is mapped by writing a sub-index between 1 and 64, the device may check whether the object specified by index /sub-index exists. If the object does not exist or the object cannot be mapped, the SDO transfer shall be aborted with the Abort SDO Transfer Service with one of the abort codes 0602 0000h or 0604 0041h.

After all objects are mapped sub-index 0 is set to the valid number of mapped objects. Finally the PDO will be created by writing to its communication parameter COB-ID. When sub-index 0 is set to a value >0 the device may validate the new PDO mapping before transmitting the response of the SDO service. If an error is detected the device shall transmit the Abort SDO Transfer Service with one of the abort codes 0602 0000h, 0604 0041h or 0604 0042h.

When sub-index 0 is read the actual number of valid mapped objects is returned.

If data types (Index 1h-7h) are mapped they serve as „dummy entries". The corresponding data in the PDO is not evaluated by the device. This optional feature is useful e.g. to transmit data to several devices using one PDO, each device only utilising a part of the PDO. It is not possible to create a dummy mapping for a TPDO.

A device that supports dynamic mapping of PDOs shall support this during the state PRE-OPERATIONAL state. If dynamic mapping during the state OPERATIONAL is supported, the SDO client is responsible for data consistency.

Figure B.17 shows the principle of PDO mapping.

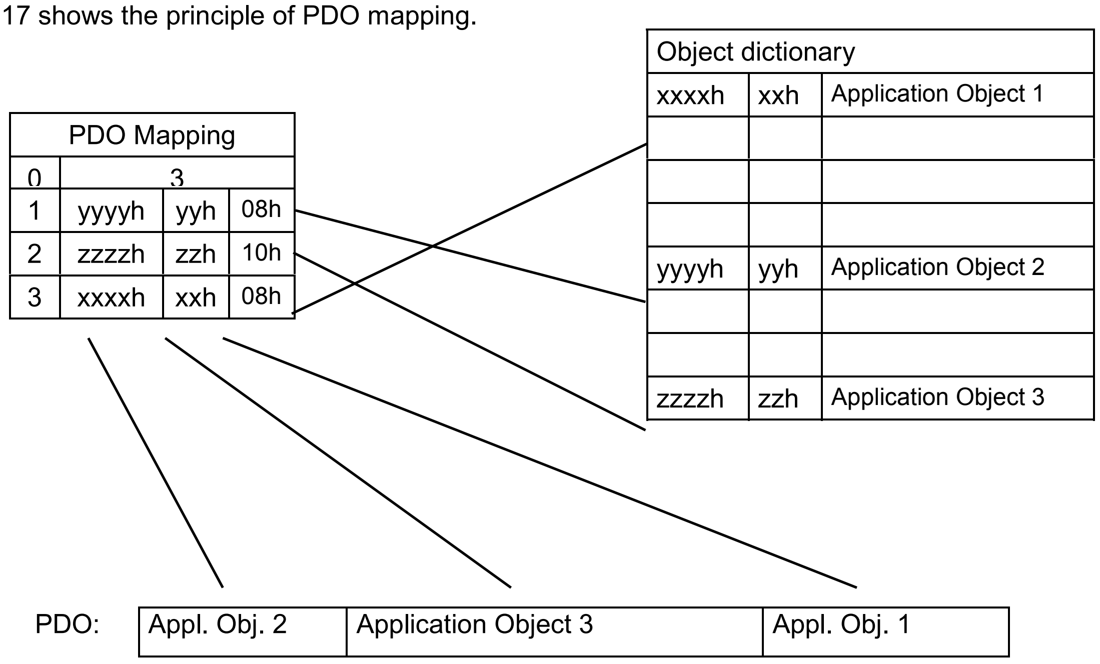


Figure B.17 - Principle of PDO mapping

OBJECT DESCRIPTION

INDEX	1600h – 17FFh
Name	receive PDO mapping
Object Code	RECORD
Data Type	PDO Mapping
Category	Conditional; Mandatory for each supported PDO

ENTRY DESCRIPTION

Sub-Index	0h
Description	number of mapped application objects in PDO
Entry Category	Mandatory
Access	ro; rw if dynamic mapping is supported
PDO Mapping	No
Value Range	0: deactivated 1 – 64: activated
Default Value	(device profile dependent)

Sub-Index	1h – 40h
Description	PDO mapping for the nth application object to be mapped
Entry Category	Conditional depends on number and size of object be mapped
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	(device profile dependent)

Object 1800h - 19FFh: Transmit PDO Communication Parameter

Shall contain the communication parameters for the PDOs the device is able to transmit. The type of the PDO communication parameter (20h) is described in annex A. A detailed description of the entries is done in the section for the Receive PDO Communication Parameter (1400h – 15FFh).

OBJECT DESCRIPTION

INDEX	1800h - 19FFh
Name	transmit PDO parameter
Object Code	RECORD
Data Type	PDO CommPar
Category	Conditional; Mandatory for each supported PDO

ENTRY DESCRIPTION

Sub-Index	0h
Description	largest sub-index supported
Entry Category	Mandatory
Access	ro
PDO Mapping	No
Value Range	2 – 5

Sub-Index	1h
Description	COB-ID used by PDO
Entry Category	Mandatory
Access	ro; rw if COB-ID may be configured
PDO Mapping	No
Value Range	UNSIGNED32 (Figure B.15)
Default Value	Index 1800h: 180h + Node-ID, Index 1801h: 280h + Node-ID, Index 1802h: 380h + Node-ID, Index 1803h: 480h + Node-ID, Index 1804h - 18FFh: disabled

Sub-Index	2h
Description	transmission type
Entry Category	Mandatory
Access	ro; rw if transmission type may be changed
PDO Mapping	No
Value Range	UNSIGNED8 (Table B.20)
Default Value	(device profile dependent)

Sub-Index	3h
Description	inhibit time
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED16
Default Value	No

Sub-Index	4h
Description	reserved
Entry Category	Optional
Access	rw
PDO Mapping	No
Value Range	UNSIGNED8
Default Value	No

Sub-Index	5h
Description	event timer
Entry Category	Optional (not used for RPDO)
Access	rw
PDO Mapping	No
Value Range	0 – not used UNSIGNED16
Default Value	No

Object 1A00h - 1BFFh: Transmit PDO Mapping Parameter

Shall contain the mapping for the PDOs the device is able to transmit. The type of the PDO mapping parameter (21h) is described in annex A. A detailed description of the entries is done in the section for the Receive PDO Mapping Parameter (1600h – 17FFh).

OBJECT DESCRIPTION

INDEX	1A00h - 1BFFh
Name	transmit PDO mapping
Object Code	RECORD
Data Type	PDO Mapping
Category	Conditional; Mandatory for each supported PDO

ENTRY DESCRIPTION

Sub-Index	0h
Description	number of mapped application objects in PDO
Entry Category	Mandatory
Access	ro; rw if dynamic mapping is supported
PDO Mapping	No
Value Range	0: deactivated 1 – 64: activated
Default Value	(device profile dependent)

Sub-Index	1h – 40h
Description	PDO mapping for the n-th application object to be mapped
Entry Category	Conditional; depends on number and size of objects to be mapped
Access	rw
PDO Mapping	No
Value Range	UNSIGNED32
Default Value	(device profile dependent)

Annex C
(informative)

Implementation recommendations

When implementing the protocols, the following rules should be obeyed to guarantee interoperability. These rules deal with the following implementation aspects:

Invalid COB's

A COB is invalid if it has a COB-ID that is used by the specified protocols, but it contains invalid parameter values according to the protocol specification. This may only be caused by errors in the lower layers or implementation errors. Invalid COB's shall be handled locally in an implementation specific way that does not fall within the scope of this specification. As far as the protocol is concerned, an invalid COB shall be ignored.

Time-out's

Since COB's may be ignored, the response of a confirmed service may never arrive. To resolve this situation, an implementation may, after a certain amount of time, indicate this to the service user (time-out). A time-out is not a confirm of that service. A time-out indicates that the service has not completed yet. The application shall deal with this situation. Time-out values are considered to be implementation specific and do not fall within the scope of this European Standard. However, it is recommended that an implementation provides facilities to adjust these time-out values to the requirements of the application.

Annex D
(informative)

Diagnostic information

Besides the application layer-transparent CAN data link layer error frames, there are several communication objects containing diagnostic information. For convenience this annex provides references where to find definitions of diagnostic information:

- Object 1001h: Error register (annex B, 5.3)
- Emergency error codes (Table 20 in 5.1.5.1)
- SDO abort codes (Table 19 in 5.1.3.3.8)
- Error control message (5.1.6.2.2)

Annex E
(informative)

Bibliography

IEEE 754:1985 Standard for binary floating-point arithmetic

BSI
389 Chiswick High Road
London
W4 4AL