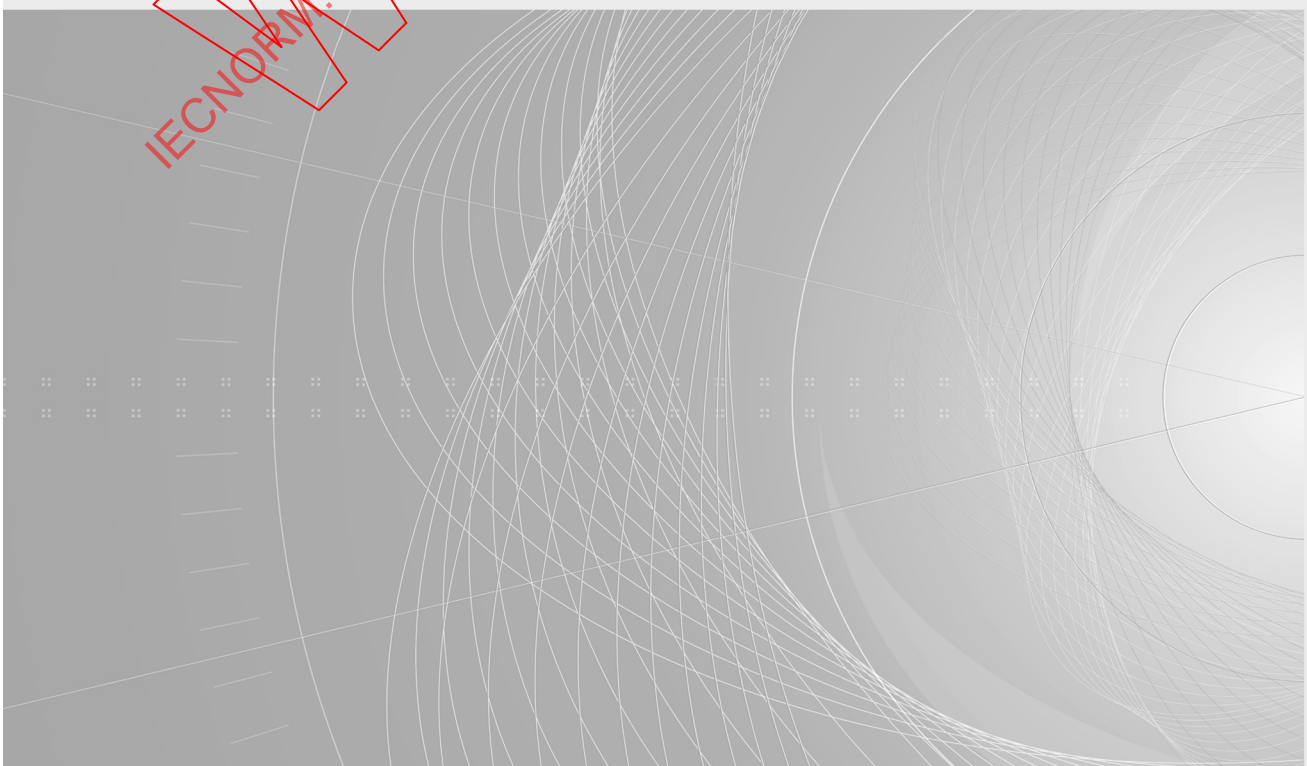
IEC 61158-6-13

Edition 1.0 2007-12

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 6-13: Application layer protocol specification – Type 13 elements



IEC 61158-6-13:2007(E)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC 61158-6-13

Edition 1.0 2007-12

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 6-13: Application layer protocol specification – Type 13 elements

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE XB

ICS 35.100.70; 25.040.40

ISBN 2-8318-9488-3

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-13: Application layer protocol specification – Type 13 elements

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

NOTE Use of some of the associated protocol Types is restricted by their intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a particular data-link layer protocol Type to be used with physical layer and application layer protocols in Type combinations as specified explicitly in the IEC 61784 series. Use of the various protocol Types in other combinations may require permission of their respective intellectual-property-right holders.

International Standard IEC 61158-6-13 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This first edition and its companion parts of the IEC 61158-6 subseries cancel and replace IEC 61158-6:2003. This edition of this part constitutes a technical addition. This part also replaces IEC/PAS 62408, published in 2005.

This edition of IEC 61158-6 includes the following significant changes from the previous edition:

a) deletion of the former Type 6 fieldbus for lack of market relevance;

b) addition of new types of fieldbuses;

c) partition of part 6 of the third edition into multiple parts numbered -6-2, -6-3, …

The text of this standard is based on the following documents:

FDIS	Report on voting
65C/476/FDIS	65C/487/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under http://webstore.iec.ch in the data related to the specific publication. At this date, the publication will be:

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

NOTE The revision of this standard will be synchronized with the other parts of the IEC 61158 series.

The list of all the parts of the IEC 61158 series, under the general title *Industrial communication networks – Fieldbus specifications*, can be found on the IEC web site.

INTRODUCTION

This part of IEC 61158 is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the "three-layer" fieldbus reference model described in IEC/TR 61158-1.

The application protocol provides the application service by making use of the services available from the data-link or other immediately lower layer. The primary aim of this standard is to provide a set of rules for communication expressed in terms of the procedures to be carried out by peer application entities (AEs) at the time of communication. These rules for communication are intended to provide a sound basis for development in order to serve a variety of purposes:

- as a guide for implementors and designers;
- for use in the testing and procurement of equipment;
- as part of an agreement for the admittance of systems into the open systems environment;
- as a refinement to the understanding of time-critical communications within OSI.

This standard is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this standard together with other standards positioned within the OSI or fieldbus reference models, otherwise incompatible systems may work together in any combination.

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-13: Application layer protocol specification – Type 13 elements

1 Scope

1.1 General

The fieldbus application layer (FAL) provides user programs with a means to access the fieldbus communication environment. In this respect, the FAL can be viewed as a "window between corresponding application programs."

This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 13 fieldbus. The term "time-critical" is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This standard specifies interactions between remote applications and defines the externally visible behavior provided by the Type 13 fieldbus application layer in terms of

a) the formal abstract syntax defining the application layer protocol data units conveyed between communicating application entities;
b) the transfer syntax defining encoding rules that are applied to the application layer protocol data units;
c) the application context state machine defining the application service behavior visible between communicating application entities;
d) the application relationship state machines defining the communication behavior visible between communicating application entities.

The purpose of this standard is to define the protocol provided to

1) define the wire-representation of the service primitives defined in IEC 61158- 5-13, and
2) define the externally visible behavior associated with their transfer.

This standard specify the protocol of the Type 13 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) and the OSI application layer structure (ISO/IEC 9545).

1.2 Specifications

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-13.

A secondary objective is to provide migration paths from previously-existing industrial communications protocols. It is this latter objective which gives rise to the diversity of protocols standardized in IEC 61158-6.

1.3 Conformance

This standard does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems.

Conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 61158-3-13, *Industrial communication networks – Fieldbus specifications - Part 3-13: Data-link layer service definition – Type 13 elements*

IEC 61158-4-13, *Industrial communication networks – Fieldbus specifications - Part 4-13: Data-link layer protocol specification – Type 13 elements*

IEC 61158-5-13, *Industrial communication networks – Fieldbus specifications - Part 5-13: Application layer service definition – Type 13 elements*

ISO/IEC 7498 (all parts), *Information technology – Open Systems Interconnection – Basic Reference Model*

ISO/IEC 8822, *Information technology – Open Systems Interconnection – Presentation service definition*

ISO/IEC 8824, *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*

ISO/IEC 9545, *Information technology – Open Systems Interconnection – Application Layer structure*

3 Terms, definitions, symbols, abbreviations and conventions

For the purposes of this document, the following terms, definitions, symbols, abbreviations and conventions apply.

3.1 ISO/IEC 7498-1 terms

This standard is partly based on the concepts developed in ISO/IEC 7498-1, and makes use of the following terms defined therein:

3.1.1
application entity

3.1.2
application process

3.1.3
application protocol data unit

3.1.4
application service element

3.1.5
application entity invocation

3.1.6
application transaction

3.1.7
transfer syntax

3.2 ISO/IEC 8822 terms

For the purposes of this document, the following term as defined in ISO/IEC 8822 applies:

3.2.1
abstract syntax

3.3 ISO/IEC 9545 terms

For the purposes of this document, the following terms as defined in ISO/IEC 9545 apply:

3.3.1
application-context

3.3.2
application-process-type

3.3.3
application-service-element

3.3.4
application control service element

3.4 ISO/IEC 8824 terms

For the purposes of this document, the following terms as defined in ISO/IEC 8824 apply:

3.4.1
any type

3.4.2
bitstring type

3.4.3
boolean type

3.4.4
choice type

3.4.5
false

3.4.6
integer type

3.4.7
module

3.4.8
null type

3.4.9
object identifier

3.4.10
octetstring type

3.4.11
production

3.4.12
simple type

3.4.13
sequence of type

3.4.14
sequence type

3.4.15
structured type

3.4.16
tag

3.4.17
tagged type

3.4.18
true

3.4.19
type

3.5 Terms and definitions from IEC 61158-5-13

3.5.1
application relationship

3.5.2
client

3.5.3
error class

3.5.4
publisher

3.5.5
server

3.5.6
subscriber

3.6 Other terms and definitions

The following terms and definitions are used in this standard:

3.6.1
receiving
service user that receives a confirmed primitive or an unconfirmed primitive, or a service provider that receives a confirmed APDU or an unconfirmed APDU

3.6.2
resource
processing or information capability of a subsystem

3.6.3
sending
service user that sends a confirmed primitive or an unconfirmed primitive, or a service provider that sends a confirmed APDU or an unconfirmed APDU

3.6.4
managing node
node that can manage the SCNM mechanism

3.6.5
controlled node
node without the ability to manage the SCNM mechanism

4 FAL syntax description

4.1 General

This description of the Type 13 abstract syntax uses formalisms similar to ASN.1, although the encoding rules differ from that standard.

4.2 FAL-AR PDU abstract syntax

4.2.1 Top level definition

```
APDU ::= CHOICE {
      [3] Isoc1
      [4] Isoc2
      [5] Asyn1
      [6] Asyn2
}
```

4.2.2 Isoc1

```
Isoc1 ::= SEQUENCE {
      message-type
      destination
      source
      reserved
      signaling-flags
      PDO-version
      reserved8
      size
      PDO-payload
}
```

4.2.3 Isoc2

```
Isoc2 ::= SEQUENCE {
      message-type
      destination
      source
      NMT-status
      signaling-flags
      PDO-version
      reserved8
      size
      PDO-payload
}
```

4.2.4 Asyn1

```
Asyn1 ::= SEQUENCE {
      message-type
      destination
      source
      NMT-status
      signaling-flags
      requested-service-ID
      requested-service-target
      EPL-version
}
```

4.2.5 Asyn2

```
Asyn2 ::= SEQUENCE {
   message-type
   destination
   source
   service-ID
   pduBody        CHOICE {
      [1h] ident-response
      [2h] status-response
      [3h] NMT-request
      [4h] NMT-command
      [5h] SDO
      [A0h...FEh] manufacturer-specific
      [FFh] reserved
   }
}
```

4.2.6 Message-type

```
message-type ::= Unsigned8                    — Contains the context specific APDU tags
```

4.2.7 Addresses

```
destination ::= Unsigned8                     — Node address (1...255)
source ::= Unsigned8                          — Node address (1...255)
```

4.2.8 Service-ID

```
service-ID ::= Unsigned8                      — Contains the context specific tags for the pduBody
```

4.2.9 Reserved8

```
reserved8 ::= Unsigned8
```

4.2.10 Reserved16

```
reserved16 ::= Unsigned16
```

4.2.11 Reserved24

```
reserved24 ::= Unsigned24
```

4.2.12 Signaling-flags

```
signaling-flags ::= BitString {
   RD             (0)
   ER             (1)
   EA             (2)
   EC             (3)
   EN             (4)
   MS             (5)
   PS             (6)
   MC             (7)
   RS_bit1        (8)
   RS_bit2        (9)
   RS_bit3        (10)
   PR_bit1        (11)
   PR_bit2        (12)
   PR_bit3        (13)
   reserved       (14)
   reserved       (15)
}
```

The different APDU types use the flags as listed in Table 1. In all cases without "x" the flags are present but not written resp. interpreted.

Table 1 – Use of signaling-flags

	Isoc1	Isoc2	Asyn1	Id.Resp.	St.Resp.
RD	x	x			
ER			x		
EA	x		x		
EC					x
EN		x			x
MS	x	x			
PS					
MC					
RS	x	x		x	x
PR	x	x		x	x

The usage of these flags is specified in IEC 61158-3-13 and IEC 61158-4-13.

4.2.13 PDO-version

PDO-version ::= Unsigned8 — High nibble: main version; low nibble: sub version

4.2.14 Size

size ::= Unsigned8 — Size of PDO payload; max. 1490 octets due to Ethernet restrictions and protocol overhead

4.2.15 PDO-payload

PDO-payload ::= Any

4.2.16 NMT-status

```
NMT-status ::= CHOICE {
   NMT_GS_OFF                    Unsigned8 ::= 0000 0000b
   NMT_xS_NOT_ACTIVE             Unsigned8 ::= 0001 1100b
   NMT_xS_PRE_OPERATIONAL_1      Unsigned8 ::= 0001 1101b
   NMT_xS_PRE_OPERATIONAL_2      Unsigned8 ::= 0101 1101b
   NMT_xS_READY_TO_OPERATE       Unsigned8 ::= 0110 1101b
   NMT_xS_OPERATIONAL            Unsigned8 ::= 1111 1101b
   NMT_xS_STOPPED                Unsigned8 ::= 0100 1101b
   NMT_xS_BASIC_ETHERNET         Unsigned8 ::= 0001 1110b
}
```

NOTE If sender = MN: "x" := "M"; if sender = CN: "x" := "C"

4.2.17 Requested-service-ID

```
requested-service-ID ::= CHOICE {
   no-service                    [0h] IMPLICIT Unsigned8
   ident-request                 [1h] IMPLICIT Unsigned8
   status-request                [2h] IMPLICIT Unsigned8
   NMT-req-inv                   [3h] IMPLICIT Unsigned8
   manufacturer-specific         [A0h]... [FEh] IMPLICIT Unsigned8
   unspecified-invite            [FFh] IMPLICIT Unsigned8
}
```

4.2.18 Requested-service-target

requested-service-target ::= Unsigned8 — Node address (1...255); not assigned (0)

4.2.19 EPL-version

EPL-version ::= Unsigned8 — High nibble: main version; low nibble: sub version

4.3 Abstract syntax of Asyn2 pduBody

4.3.1 Ident-response

4.3.1.1 Overview

```
Ident-response ::= SEQUENCE {
   signaling-flags
   NMT-status
   reserved8
   EPL-version
   reserved8
   feature-flags                BitString          — (see 4.3.1.2)
   MTU                          Unsigned16         — size of the largest possible IP frame incl. header
   poll-in-size                 Unsigned16         — actual CN setting for Isoc1 data block size
   poll-out-size                Unsigned16         — actual CN setting for Isoc2 data block size
   response-time                Unsigned32         — time required by the CN to respond to Isoc1
   reserved16
   device-type                  Unsigned32         — CN's device type
   vendor-ID                    Unsigned32         — CN's vendor ID
   product-code                 Unsigned32         — CN's product code
   revision-number              Unsigned32         — CN's revision number
   serial-number                Unsigned32         — CN's serial number
   vendor-specific-extension-1  Unsigned64         — for vendor specific purpose, to be filled with zeros if not used
   verify-configuration-date    Unsigned32         — CN's configuration date
   verify-configuration-time    Unsigned32         — CN's configuration time
   application-sw-date          Unsigned32         — CN's application software date
   application-sw-time          Unsigned32         — CN's application software time
   IP-address                   Unsigned32         — current IP address value of the CN
   subnet-mask                  Unsigned32         — current IP subnet mask of the CN
   default-gateway              Unsigned32         — current IP default gateway of the CN
   host-name                    VisibleString32    — current DNS host name of the CN
   vendor-specific-extension-2  SEQUENCE SIZE(64) OF Unsigned64
                                                   — for vendor specific purpose, to be filled with zeros if not in use
}
```

NOTE Some of the above listed elements are sometimes summarized as follows:
"poll-in-size" through "response-time" are summarized under the term "cycle-timing",
"device-type" through "serial-number" under "identity",
"verify-configuration-date" and "verify-configuration-time" under "verify-configuration",
"application-sw-date" and "application-sw-time" under "application-software-version",
"vendor-specific-extension-1" and "vendor-specific-extension-2" under "vendor-specific-extensions",
"IP-address" through "default-gateway" under "IP-address",

4.3.1.2 Feature-flags

```
feature-flags ::= BitString {
   Isochronous                  (0)        — device may be isochronously accessed via Isoc1
   SDO by UDP/IP                (1)        — device supports SDO communication via UDP/IP
   SDO by ASnd                  (2)        — device supports SDO communication via ASnd
   reserved for future use      (3)
   NMT-info services            (4)        — device supports NMT Info Services
   Extended NMT-state-commands  (5)        — device supports Extended NMT State Commands
   Dynamic PDO mapping          (6)        — device supports dynamic PDO Mapping
   NMT services by UDP/IP       (7)        — device supports NMT Services by UDP/IP
   Configuration manager        (8)        — device supports Configuration Manager functions
   Multiplexed access           (9)        — CN device supports multiplexed isochronous access.
   Node-ID setup by SW          (10)       — device supports NodeID setup by software
   MN basic ethernet mode       (11)       — MN device supports Basic Ethernet Mode
   Routing Type 1 support       (12)       — device supports Routing Type 1 functions
   Routing Type 2 support       (13)       — device supports Routing Type 2 functions
   reserved_bit1 through bit18  (13)...(31)
}
```

4.3.2 Status-response

4.3.2.1 Overview

```
Status-response ::= SEQUENCE {
   signaling-flags
   NMT-status
   reserved24
   static-error-bit-field
   List-of-errors
}
```

4.3.2.2 Static-error-bitf ield

```
static-error-bit-field ::= SEQUENCE {
   error-register          BitString                          — (see 4.3.2.3)
   reserved8
   specific-errors         SEQUENCE SIZE (n) OF Unsigned8     — Device profile or vendor specific errors
                                                                n depends on device configuration
}
```

4.3.2.3 Error-register

```
error-register ::= BitString {
   Generic error           (0)                                — 0, if no static error present, else 1
   Current                 (1)      OPTIONAL
   Vorltage                (2)      OPTIONAL
   Temperature             (3)      OPTIONAL
   Communication error     (4)      OPTIONAL
   Device profile specific (5)      OPTIONAL
   reserved                (6)      OPTIONAL                  — always 0
   Manufacturer specific   (7)      OPTIONAL
}
```

4.3.2.4 List-of-errors

```
List-of-errors ::= SEQUENCE SIZE (k) OF ErrorEntry          — k :>= 2, depends on device configuration
```

4.3.2.5 ErrorEntry

```
ErrorEntry ::= SEQUENCE {
   error-type              Unsigned16                         — (see 4.3.2.6)
   error-code              Unsigned16                         — (see 4.3.2.6 and A.3)
   time-stamp              Unsigned64
   additional-information  Unsigned64
}
```

4.3.2.6 Error-type

The possible values in error-type and their meaning are listed in Table 2.

Table 2 – Values of error-type

Octet	Bit	Value	Description
0 .. 1	15 (status)	0b	ERR_History_ADOM Entry
		1b	Status entry in Status-response frame (Bit 14 shall be set to 0b)
	14 (send)	0b	ERR_History_ADOM only
		1b	Additional to the ERR_History_ADOMthe entry shall also be entered in to the emergency queue of the error signaling
	13 .. 12 (mode)	0h	Not allowed in ERR_History_ADOM. Entries with this mode may only be used by the error signaling itself to indicate the termination of the history entries in the Status-response frame
		1h	An error has occurred and is active (e.g. short circuit of output detected)
		2h	An active error was cleared (e.g. no short circuit anymore) (not allowed for status entries)
		3h	An error / event occurred (not allowed for status entries)
	11 .. 0 (profile)	000h	Reserved
		001h	error-code contains a vendor specific error code
		002h	error-code contains Type 13 network specific communication profile errors (see A.3)
		003h .. FFFh	error-code contains device profile specific errors

4.3.3 NMT-request

```
NMT-request ::= SEQUENCE {
   NMT-requested-command-ID          Unsigned8      — value range see A.2
   NMT-requested-command-target      Unsigned8      — target node address
   NMT-requested-command-data        Any
}
```

4.3.4 NMT-command

```
NMT-command ::= SEQUENCE {
   NMT-command-ID  CHOICE {                         — value range see A.2
      NMT-state-command                Unsigned8
      NMT-info                         Unsigned8
   }
   reserved8
   NMT-command-data   CHOICE {
      date-time                        TimeOfDay     — only if NMT-command-ID = B0h; see A.4
      node-states      SEQUENCE SIZE (255) OF Unsigned8 — if NMT-command-ID = 96h; see A.4;
                                                      reports each node state individually
      EPL-node-list                                 — if NMT-command-ID = 40h...95h, A0h;
                                                      see A.4
      NULL                                          — else
   }
}
```

4.3.5 SDO

4.3.5.1 Overview

```
SDO ::= SEQUENCE {
   SequenceLayer
   CommandLayer
}
```

4.3.5.2 SequenceLayer

```
SequenceLayer ::= SEQUENCE {
   SequenceFlags     Bitstring {
      rcon_bit1                        (0)      — 0: no connection; 1: initialisation; 2: connection valid
      rcon_bit2                        (1)          3: error response (retransmission requested)
      rsnr_bit1                        (2)      — 0..63
      rsnr_bit2                        (3)
      rsnr_bit3                        (4)
      rsnr_bit4                        (5)
      rsnr_bit5                        (6)
      rsnr_bit6                        (7)
      scon_bit1                        (8)      — 0: no connection; 1: initialisation; 2: connection valid
      scon_bit2                        (9)          3: connection valid with acknowledge request
      ssnr_bit1                        (10)     — 0..63
      ssnr_bit2                        (11)
      ssnr_bit3                        (12)
      ssnr_bit4                        (13)
      ssnr_bit5                        (14)
      ssnr_bit6     (15)
   }
   reserved16
}
```

4.3.5.3 CommandLayer

```
CommandLayer ::= SEQUENCE {
   SDOCommandHeader          SEQUENCE {
      reserved8
      invoke-ID
      segmentation_abort_response
      command-ID
      segment-size
      reserved16
   }
   SDO-command      CHOICE {
      write-by-index-request           [1h] IMPLICIT WriteByIndex-RequestPDU
      read-by-index-request            [2h] IMPLICIT ReadByIndex-RequestPDU
      write-all-by-index-request       [3h] IMPLICIT WriteAllByIndex-RequestPDU
      read-all-by-index-request        [4h] IMPLICIT ReadAllByIndex-RequestPDU
      write-multiple-by-index-request  [31h] IMPLICIT WriteMultipleByIndex-RequestPDU
      read-multiple-by-index-request   [32h] IMPLICIT ReadMultipleByIndex-RequestPDU
      abort                            IMPLICIT AbortPDU
      write-by-index-response          [1h] IMPLICIT WriteByIndex-ResponsePDU
      read-by-index-response           [2h] IMPLICIT ReadByIndex-ResponsePDU
      write-all-by-index-response      [3h] IMPLICIT WriteAllByIndex-ResponsePDU
      read-all-by-index-response       [4h] IMPLICIT ReadAllByIndex-ResponsePDU
      write-multiple-by-index-response [31h] IMPLICIT WriteMultipleByIndex-ResponsePDU
      read-multiple-by-index-response  [32h] IMPLICIT ReadMultipleByIndex-ResponsePDU
   }
}
```

4.3.5.4 Invoke ID

```
invoke-ID ::= Unsigned8
```

4.3.5.5 Segmentation_abort_response

```
segmentation_abort_response ::= BitString {
   reserved                            (0)
   reserved                            (1)
   reserved                            (2)
   reserved                            (3)
   segmentation_bit1                   (4)      — 0: Expedited transfer; 1: initiate segment transfer
   segmentation_bit2                   (5)          2: segment; 3: segment transfer complete
   abort                               (6)      — 0: transfer ok; 1: abort
   response                            (7)      — 0: request; 1: response
}
```

4.3.5.6 Command-ID

```
command-ID ::= Unsigned8                 —  Contains context specific tag for SDO-command
```

4.3.5.7 Segment-size

segment-size::= Unsigned16 — Length of segment data. Counting from the beginning of the "SDO-command". Valid value range: 0...1458

4.3.5.8 WriteByIndex services

```
WriteByIndex-RequestPDU ::= SEQUENCE {
   data-size                          —  only present if in SDOCommandHeader segmentation = "initiate"
   index
   sub-index
   reserved8
   payload-data
}

WriteByIndex-ResponsePDU ::= NULL
```

4.3.5.9 ReadByIndex services

```
ReadByIndex-RequestPDU ::= SEQUENCE {
   index
   sub-index
   reserved16
}

ReadByIndex-ResponsePDU ::= SEQUENCE {
   data-size                          —  only present if in SDOCommandHeader segmentation = "initiate"
   payload-data
}
```

4.3.5.10 WriteAllByIndex services

```
WriteAllByIndex-RequestPDU ::= SEQUENCE {
   data-size                          —  only present if in SDOCommandHeader segmentation = "initiate"
   index
   reserved16
   payload-data
}

WriteAllByIndex-ResponsePDU ::= NULL
```

4.3.5.11 ReadAllByIndex services

```
ReadAllByIndex-RequestPDU ::= SEQUENCE {
   index
   reserved16
}

ReadAllByIndex-ResponsePDU ::= SEQUENCE {
   data-size                          —  only present if in SDOCommandHeader segmentation = "initiate"
   payload-data
}

WriteByName-ResponsePDU ::= NULL
```

4.3.5.12 WriteMultipleByIndex services

```
WriteMultipleByIndex-RequestPDU ::= SEQUENCE {
   data-size                          —  only present if in SDOCommandHeader segmentation = "initiate"
   offset (k)                         —  Byte offset of next payload data set
   index
   sub-index
   padding-length
   payload-data
   offset (m)                         —  Byte offset of next payload data set
   index
   sub-index
   padding-length
   payload-data
   …                                  —  (further write requests)
}

WriteMultipleByIndex-ResponsePDU ::= SEQUENCE {
   data-size                          —  only present if in SDOCommandHeader segmentation = "initiate"
   index
   sub-index
   sub-abort
   sub-abort-code
   index
   sub-index
   sub-abort
   sub-abort-code
   …                                  —  (further write responses)
}
```

4.3.5.13 ReadMultipleByIndex services

```
ReadMultipleByIndex-RequestPDU ::= SEQUENCE {
   data-size                          —  only present if in SDOCommandHeader segmentation = "initiate"
   index
   sub-index
   reserved8
   index
   sub-index
   reserved8
   …                                  —  (further read requests)
}

ReadMultipleByIndex-ResponsePDU ::= SEQUENCE {
   data-size                          —  only present if in SDOCommandHeader segmentation = "initiate"
   offset (k)                         —  offset of the next payload data set
   index
   sub-index
   sub-abort -padding-length
   payload-data / sub-abort-code
   offset (m)                         —  offset of the next payload data set
   index
   sub-index
   sub-abort-padding-length
   payload-data / sub-abort-code
   …                                  —  (further read requests)
}
```

4.3.5.14 AbortPDU

```
AbortPDU ::= {
   abort-code     Unsigned32          —  see A.1 for valid values
}
```

4.3.5.15 Data size

data-size ::= Unsigned32 — Length of transferred data block. Counting from the beginning of the "SDO-command". Valid value range: 0…2^{32}-1.

4.3.5.16 Index

index ::= Unsigned16 — Specifies an entry of the device object dictionary; 0.. 65.535

4.3.5.17 Sub-index

sub-index ::= Unsigned8 — Specifies a component of a device object dictionary entry; 0..254

4.3.5.18 Payload-data

payload-data ::= Any — application dependent type and length;
total frame length must comply with Ethernet rules

4.3.5.19 Offset

offset (i) ::= Unsigned8 — offset of specified data; i is 4-aligned

4.3.5.20 Padding-length

padding-length ::= Unsigned8 — Number of padding bytes in the last quadlet (4-byte word) of the payload data; coded in the two least significant bits

4.3.5.21 Sub-abort

sub-abort ::= Unsigned8 — 0: transfer ok; 1: abort; coded in the most significant bit

4.3.5.22 Sub-abort-padding-length

sub-abort-padding-length ::= Unsigned8 — sub-abort (see 4.3.5.21) and padding-length (see 4.3.5.20) merged in one octet.

4.3.5.23 Sub-abort-code

sub-abort-code ::= Unsigned32 — Values and meaning identically equal to abort-code, see A.1.

4.3.6 Manufacturer-specific

These parts are reserved for manufacturer specific purpose. Their specification is not in the scope of this international standard.

5 Transfer syntax

5.1 Encoding of data types

5.1.1 General description of data types and encoding rules

To be able to exchange meaningful data, the format of this data and its meaning have to be known by the producer and consumer(s). This specification models this by the concept of data types.

The encoding rules define the representation of values of data types and the transfer syntax for the representations. Values are represented as bit sequences. Bit sequences are transferred in sequences of octets (bytes). For numerical data types the encoding is little endian style as shown in Table 3.

5.1.2 Transfer syntax for bit sequences

For transmission a bit sequence is reordered into a sequence of octets. Hexadecimal notation is used for octets as specified in ISO/IEC 9899. Let $b = b_0 \ldots b_{n-1}$ be a bit sequence. 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 Table 3. The bits b_i, $i \geq n$ of the highest numbered octet are do not care bits.

Table 3 – Transfer syntax for bit sequences

octet number	1.	2.	k.
	5.1.3 b_7 .. b_0	**5.1.4 b_{15} .. b_8**	**5.1.5 b8k –1 .. b8k -8**

Octet 1 is transmitted first and octet *k* is transmitted last. The bit sequence is transferred as follows across the network (transmission order within an octet is determined by ISO/IEC 8802-3):

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

EXAMPLE

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

The bit sequence $b = b_0 .. b_9 = 0011\ 1000\ 01_b$ represents an Unsigned10 with the value 21Ch and is transferred in two octets: First 1Ch and then 02h.

5.1.6 Encoding of a Boolean value

Data of basic data type BOOLEAN attains the values TRUE or FALSE.

The values are represented as bit sequences of length 1. The value TRUE is represented by the bit sequence 1, and FALSE by 0.

A BOOLEAN shall be transferred over the network as UNSIGNED8 of value 1 (TRUE) resp. 0 (FALSE). Sequent BOOLEANs may be packed to one UNSIGNED8. Sequences of BOOLEAN and BIT type items may be also packed to one UNSIGNED8.

5.1.7 Encoding of an Unsigned Integer value

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

The bit sequence

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

is assigned the value

$$UNSIGNEDn(b) = b_{n-1}\times 2^{n-1} + ... + b_1\times 2^1 + b_0\times 2^0$$

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

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

The following UNSIGNEDn data types are transferred as shown in Table 4.

Table 4 – Transfer syntax for data type UNSIGNEDn

octet number	0	1	2	3	4	5	6	7
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	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

The data types UNSIGNED24, UNSIGNED40, UNSIGNED48 and UNSIGNED56 shall not be applied by new applications.

UNSIGNEDn data types of length deviating from the values listed above may be applied by compound data types only.

5.1.8 Encoding of a Signed Integer

Data of basic data type INTEGERn has values in the integers. The value range is from -2^{n-1} to $2^{n-1}-1$. The data is represented as bit sequences of length n. The bit sequence

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

is assigned the value

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

and, performing two's complement arithmetic,

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

Note that the bit sequence starts on the left with the least significant bit.

Example: The value $-266_d = 0xFEF6_h$ with data type Integer16 is transferred in two octets, first 0xF6 and then 0xFE.

The INTEGERn data types are transferred as specified in Table 5.

Table 5 – Transfer syntax for data type Integern

octet number	0	1	2	3	4	5	6	7
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	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

The data types INTEGER24, INTEGER40, INTEGER48 and INTEGER56 should not be applied by new applications.

INTEGERn data types of length deviating from the values listed above may be applied by compound data types only.

5.1.9 Encoding of a Floating point value

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

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

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

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

The bit sequence

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

is assigned the value (finite non-zero number)

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

Here

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

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

$F = 2^{-23} \times (b_{22} \times 2^{22} + \ldots + b_1 \times 2^1 + b_0 \times 2^0)$ 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 starts on the left with the least significant bit.

5.1.10 Encoding of an Octet String value

The data type OCTET_STRINGlength is defined as follows; "length" is the length of the octet string.

ARRAY [length] OF UNSIGNED8 OCTET_STRINGlength

5.1.11 Encoding of a Visible String value

VISIBLE_CHAR are 0_h and the range from 20_h to $7E_h$. The data are interpreted as ISO 646-1973(E) 7- bit coded characters. "length" is the length of the visible string.

UNSIGNED8 VISIBLE_CHAR

ARRAY [length] OF VISIBLE_CHAR VISIBLE_STRINGlength

There is no 0_h necessary to terminate the string.

5.1.12 Encoding of a Unicode String Value

The data type UNICODE_STRINGlength is defined below; "length" is the length of the unicode string.

ARRAY [length] OF UNSIGNED16 UNICODE_STRINGlength

5.1.13 Encoding of a Time of Day value

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

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

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

The encoding is as shown in Figure 1.

bits	7	6	5	4	3	2	1	0	
octets									
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	number of milliseconds since midnight
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	number of days since 01.01.84
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
	msb								

Figure 1 – Encoding of Time of Day value

5.1.14 Encoding of a Time Difference value

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

Time differences are sums of numbers of days and milliseconds. Component "ms" is the number milliseconds. Component "days" is the number of days.

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

The encoding is as shown in Figure 2.

bits	7	6	5	4	3	2	1	0	
octets									
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	milliseconds
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	days
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
	msb								

Figure 2 – Encoding of Time Difference value

6 FAL protocol state machines

Interface to FAL services and protocol machines are specified in this subclause.

NOTE The state machines specified in this subclause and ARPMs defined in the following clauses only define the valid events for each. It is a local matter to handle invalid events.

The behavior of the FAL is described by the protocol machines shown in Figure 3. Specific sets of these protocol machines are defined for different AREP types. Figure 3 also shows the primitives exchanged between the protocol machines.

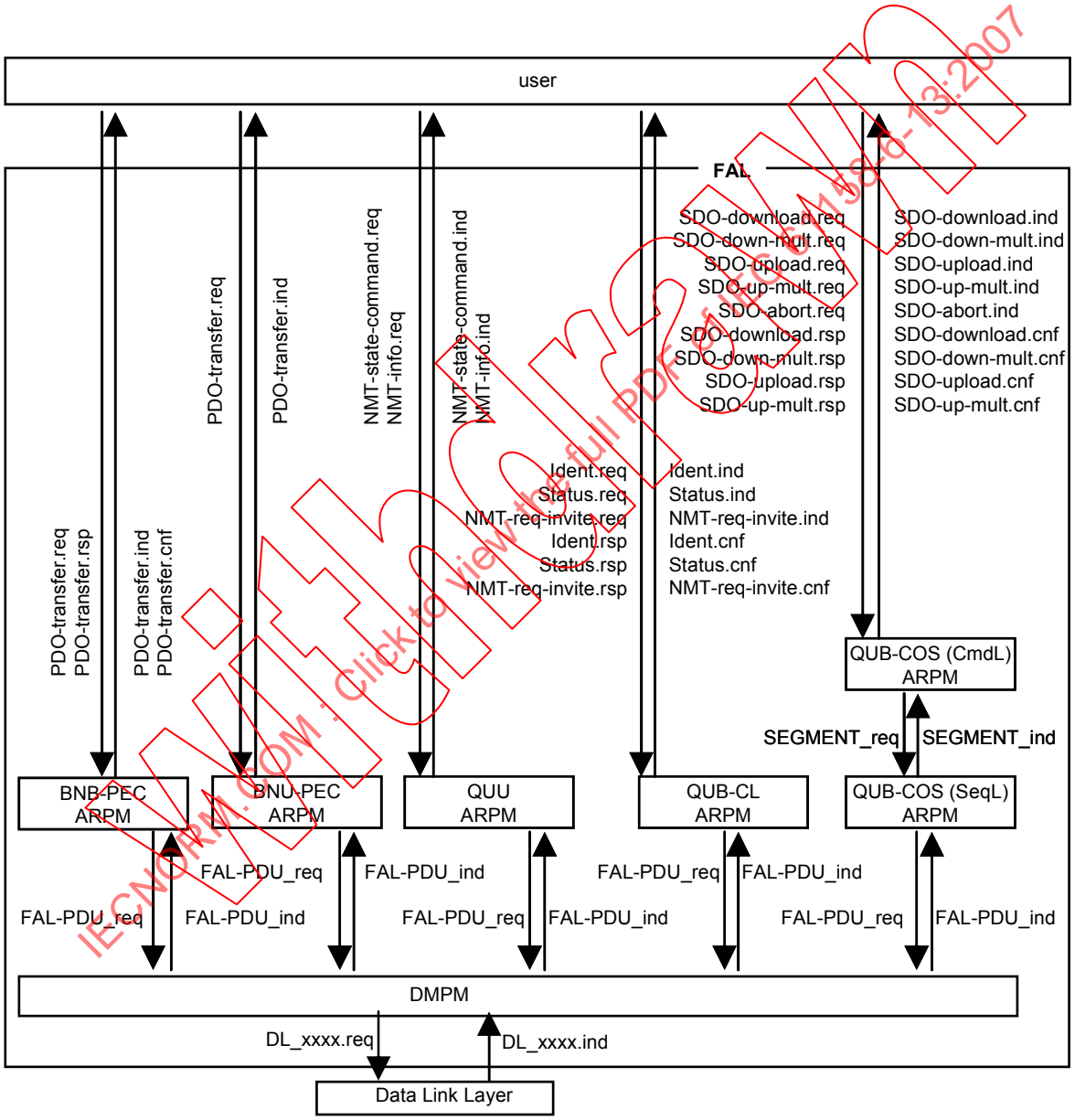


Figure 3 – Primitives exchanged between protocol machines

7 AP context state machine

There is no AP-context state machine defined for this protocol.

8 FAL service protocol machine

There is no FAL service protocol state machine defined for this protocol.

9 AR protocol machine

9.1 Buffered-network-scheduled bi-directional pre-established connection (BNB-PEC) ARPM

9.1.1 BNB-PEC primitive definitions

9.1.1.1 Primitives exchanged between BNB-PEC ARPM and user

Table 6 and Table 7 list the primitives exchanged between the ARPM and the user.

Table 6 – Primitives issued by user to BNB-PEC ARPM

Primitive name	Source	Associated parameters	Functions
PDO-transfer.req	user	AREP PDO PDO-version	Refer to service data definitions in IEC 61158-5-13
PDO-transfer.rsp	user	AREP PDO PDO-version	Refer to service data definitions in IEC 61158-5-13

Table 7 – Primitives issued by BNB-PEC ARPM to user

Primitive name	Source	Associated parameters	Functions
PDO-transfer.ind	ARPM	AREP PDO PDO-version	Refer to service data definitions in IEC 61158-5-13
PDO-transfer.cnf	ARPM	AREP PDO PDO-version	Refer to service data definitions in IEC 61158-5-13

9.1.1.2 Parameters of primitives

The parameters of the primitives are described in IEC 61158-5-13.

9.1.2 DLL mapping of BNB-PEC class

9.1.2.1 Formal model

This subclause describes the mapping of the BNB-PEC AREP class to the Type 13 data link layer defined in IEC 61158-3-13 and IEC 61158-4-13. It does not redefine the DLSAP

attributes or DLME attributes that are or will be defined in the data link layer standard; rather, it defines how they are used by this AR class.

NOTE A means to configure and monitor the values of these attributes are not in the scope of this International Standard.

The DLL mapping attributes and their permitted values and the DLL services used with the BNB-PEC AREP class are defined in this subclause.

CLASS:			**Type 13 BNB-PEC**
PARENT CLASS:			**Buffered network-scheduled bi-directional pre-established connection AREP**
ATTRIBUTES:			
1	(m)	KeyAttribute:	LocalDlcepAddress
2	(m)	Attribute:	RemoteDlcepAddress
DLL SERVICES:			
1	(m)	OpsService:	DL-PDO

9.1.2.2 Attributes

LocalDlcepAddress
This attribute specifies the local DLCEP address and identifies the DLCEP. The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

RemoteDlcepAddress
This attribute specifies the remote DLCEP address and identifies the DLCEP.

9.1.2.3 DLL services

Refer to IEC 61158-3-13 for DLL service descriptions.

9.1.3 BNB-PEC ARPM state machine

9.1.3.1 BNB-PEC ARPM states

The BNB-PEC ARPM state machine has only one state called "ACTIVE", see Figure 4.

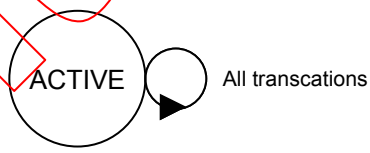


Figure 4 – State transition diagram of BNB-PEC ARPM

9.1.3.2 BNB-PEC ARPM state table

Table 8 and Table 9 define the state machine of the BNB-PEC ARPM.

Table 8 – BNB-PEC ARPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	ACTIVE	PDO-transfer.req
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type :="lsoc1"
		data := PDO-transfer.req)
		}
			ACTIVE

S2	ACTIVE	PDO-transfer.rsp
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type :="lsoc2"
		data := PDO-transfer.rsp)
		}
			ACTIVE

NOTE Transactions S1 is executed by the MN only, transaction S2 is executed by the addressed CN only.			

Table 9 – BNB-PEC ARPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state
R1	ACTIVE	FAL-PDU_ind && message-type = "lsoc1" ⇒ PDO-transfer.ind	ACTIVE
R2	ACTIVE	FAL-PDU_ind && message-type = "lsoc2" ⇒ PDO-transfer.cnf	ACTIVE
NOTE Transaction R1 is executed by the CNs only, transaction R2 is executed by the MN and may be executed by CNs depending on their configuration.			

9.1.3.3 Functions used by BNB-PEC ARPM

The receipt of a FAL-PDU_ind primitive is always followed by its decoding to derive its relevant parameters for the state machine. Thus this implicit function is not listed separately.

Table 10 defines the other function used by this state machine.

Table 10 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
message-type data additional information		dlsdu	
Function			
Builds a FAL-PDU out of the parameters given as input variables.			

9.2 Buffered-network-scheduled uni-directional pre-established connection (BNU-PEC) ARPM

9.2.1 BNU-PEC primitive definitions

9.2.1.1 Primitives exchanged between BNU-PEC ARPM and user

Table 11 and Table 12 list the primitives exchanged between the ARPM and the user.

Table 11 – Primitives issued by user to BNU-PEC ARPM

Primitive name	Source	Associated parameters	Functions
PDO-transfer.req	user	AREP PDO PDO-version	Refer to service data definitions in IEC 61158-5-13

Table 12 – Primitives issued by BNU-PEC ARPM to user

Primitive name	Source	Associated parameters	Functions
PDO-transfer.ind	ARPM	AREP PDO PDO-version	Refer to service data definitions in IEC 61158-5-13

9.2.1.2 Parameters of primitives

The parameters of the primitives are described in IEC 61158-5-13.

9.2.2 DLL mapping of BNU-PEC class

9.2.2.1 Formal model

This subclause describes the mapping of the BNU AREP class to the Type 13 data link layer defined in IEC 61158-3-13 and IEC 61158-4-13. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the data link layer standard; rather, it defines how they are used by this AR class.

NOTE A means to configure and monitor the values of these attributes are not in the scope of this International Standard.

The DLL mapping attributes and their permitted values and the DLL services used with the BNU AREP class are defined in this subclause.

CLASS: **Type 13 BNU-PEC**

PARENT CLASS: **Buffered network-scheduled uni-directional pre-established connection AREP**

ATTRIBUTES:

1 (m) KeyAttribute: PublisherDlcepAddress

DLL SERVICES:

1 (m) OpsService: DL-PDO

9.2.2.2 Attributes

PublisherDlcepAddress
This attribute specifies the publisher's DLCEP address and identifies the DLCEP. The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

9.2.2.3 DLL services

Refer to IEC 61158-3-13 for DLL service descriptions.

9.2.3 BNU-PEC ARPM state machine

9.2.3.1 BNU-PEC ARPM states

The BNU-PEC ARPM state machine has only one state called "ACTIVE", see Figure 5.

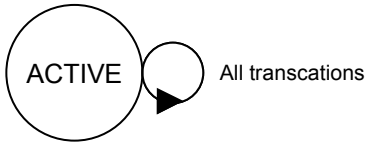


Figure 5 – State transition diagram of BNU-PEC ARPM

9.2.3.2 BNU-PEC ARPM state table

Table 13 and Table 14 define the state machine of the BNU-PEC ARPM.

Table 13 – BNU-PEC ARPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	ACTIVE	PDO-transfer.req
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type :="Isoc2"
		data := PDO-transfer.req)
		}
			ACTIVE

NOTE This transaction is executed by the MN only.			

Table 14 – BNU-PEC ARPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state
R1	ACTIVE	FAL-PDU_ind && message-type = "Isoc2" ⇒ PDO-transfer.ind	ACTIVE
NOTE This transaction is executed by the CNs only.			

9.2.3.3 Functions used by BNU ARPM

The receipt of a FAL-PDU_ind primitive is always followed by its decoding to derive its relevant parameters for the state machine. Thus this implicit function is not listed separately.

Table 15 defines the other function used by this state machine.

Table 15 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
message-type data additional information		dlsdu	
Function			
Builds a FAL-PDU out of the parameters given as input variables.			

9.3 Queued user-triggered uni-directional (QUU) ARPM

9.3.1 QUU primitive definitions

9.3.1.1 Primitives exchanged between QUU ARPM and user

Table 16 and Table 17 list the primitives exchanged between the ARPM and the user.

Table 16 – Primitives issued by user to QUU ARPM

Primitive name	Source	Associated parameters	Functions
NMT-state-command.req	user	AREP command-ID node-list	Refer to service data definitions in IEC 61158-5-13.
NMT-info.req	user	AREP publish-node-list publish-time	Refer to service data definitions in IEC 61158-5-13.

Table 17 – Primitives issued by QUU ARPM to user

Primitive name	Source	Associated parameters	Functions
NMT-state-command.ind	ARPM	AREP command-ID node-list	Refer to service data definitions in IEC 61158-5-13.
NMT-info.ind	ARPM	AREP publish-node-list publish-time	Refer to service data definitions in IEC 61158-5-13.

9.3.1.2 Parameters of primitives

The parameters of the primitives are described in IEC 61158-5-13.

9.3.2 DLL mapping of QUU AREP class

9.3.2.1 Formal model

This subclause describes the mapping of the QUU AREP class to the Type 13 data link layer defined in IEC 61158-3-13 and IEC 61158-4-13. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the data link layer standard; rather, it defines how they are used by this AR class.

NOTE A means to configure and monitor the values of these attributes are not in the scope of this International Standard.

The DLL Mapping attributes and their permitted values and the DLL services used with the QUU AREP class are defined in this subclause.

CLASS: **Type 13 QUU**
PARENT CLASS: **Queued User-triggered uni-directional AREP**
ATTRIBUTES:
1 (m) KeyAttribute: LocalDlcepAddress
2 (m) Attribute: RemoteDlcepAddress
DLL SERVICES:
1 (m) OpsService: DL-CMD

9.3.2.2 Attributes

LocalDlcepAddress
This attribute specifies the local DLCEP address and identifies the DLCEP. The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

RemoteDlcepAddress

This attribute specifies the remote DLCEP address and identifies the DLCEP

9.3.2.3 DLL services

Refer to IEC 61158-3-13 for DLL service descriptions.

9.3.3 QUU ARPM state machine

9.3.3.1 QUU ARPM states

The QUU ARPM state machine has only one state called "ACTIVE", see Figure 6.

Figure 6 – State transition diagram of QUU ARPM

9.3.3.2 QUU ARPM state table

Table 18 and Table 19 define the state machine of the QUU ARPM.

Table 18 – QUU ARPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	ACTIVE	NMT-state-command.req
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 6h — "Asyn2"
		service-id := 4h — "NMT-command"
		data := NMT-state-command.req)
		}
			ACTIVE

S2	ACTIVE	NMT-info.req
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 6h — "Asyn2"
		service-id := 4h — "NMT-command"
		data := NMT-info.req)
		}
			ACTIVE

NOTE These transactions are executed by the MN only.			

Table 19 – QUU ARPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state
R1	ACTIVE	FAL-PDU_ind && service-id = 4h — "NMT-command" && 20h <= command-ID <= 5Fh — (see A.2) ⇒ NMT-state-command.ind	ACTIVE
R2	ACTIVE	FAL-PDU_ind && service-id = 4h — "NMT-command" && 80h <= command-ID <= BFh — (see A.2) ⇒ NMT-info.ind	ACTIVE
NOTE These transactions are executed by the CNs only.			

9.3.3.3 Functions used by QUU ARPM

The receipt of a FAL-PDU_ind primitive is always followed by its decoding to derive its relevant parameters for the state machine. Thus this implicit function is not listed separately.

Table 20 defines the other functions used by this state machine.

Table 20 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
message-type service-id message-type data additional information		dlsdu	
Function			
Builds a FAL-PDU out of the parameters given as input variables.			

9.4 Queued user-triggered bi-directional connectionless (QUB-CL) ARPM

9.4.1 QUB-CL Primitive definitions

9.4.1.1 Primitives exchanged between QUB-CL ARPM and user

Table 21 and Table 22 list the primitives exchanged between the ARPM and the user.

Table 21 – Primitives issued by user to QUB-CL ARPM

Primitive name	Source	Associated parameters	Functions
Ident.req	user	AREP	Refer to service data definitions in IEC 61158-5-13
Status.req	user	AREP	Refer to service data definitions in IEC 61158-5-13
NMT-req-invite.req	user	AREP	Refer to service data definitions in IEC 61158-5-13
Ident.rsp	user	AREP NMT-status EPL-version feature-flags cycle-timing Identity verify-configuration application-software-version IP-address host-name vendor-specific-extensions	Refer to service data definitions in IEC 61158-5-13
Status.rsp	user	AREP NMT-status static-error error-history	Refer to service data definitions in IEC 61158-5-13
NMT-req-invite.rsp	user	AREP command-ID target-node data	Refer to service data definitions in IEC 61158-5-13

Table 22 – Primitives issued by QUB-CL ARPM to user

Primitive name	Source	Associated parameters	Functions
Ident.ind	ARPM	AREP	Refer to service data definitions in IEC 61158-5-13
Status.ind	ARPM	AREP	Refer to service data definitions in IEC 61158-5-13
NMT-req-invite.ind	ARPM	AREP	Refer to service data definitions in IEC 61158-5-13
Ident.cnf	ARPM	AREP NMT-status EPL-version feature-flags cycle-timing Identity verify-configuration application-software-version IP-address host-name vendor-specific-extensions	Refer to service data definitions in IEC 61158-5-13
Status.cnf	ARPM	AREP NMT-status static-error error-history	Refer to service data definitions in IEC 61158-5-13
NMT-req-invite.cnf	ARPM	AREP command-ID target-node data	Refer to service data definitions in IEC 61158-5-13

9.4.1.2 Parameters of primitives

The parameters of the primitives are described in IEC 61158-5-13.

9.4.2 DLL mapping of QUB-CL AREP class

9.4.2.1 Formal model

This subclause describes the mapping of the QUB-CL AREP class to the Type 13 data link layer defined in IEC 61158-3-13 and IEC 61158-4-13. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the data link layer standard; rather, it defines how they are used by this AR class.

NOTE A means to configure and monitor the values of these attributes are not in the scope of this standard.

The DLL Mapping attributes and their permitted values and the DLL services used with the QUB-CL AREP class are defined in this subclause.

CLASS:			**Type 13 QUB-CL**
PARENT CLASS:			**Queued User-triggered Bi-directional connectionless AREP**
ATTRIBUTES:			
1	(m)	KeyAttribute:	LocalDlcepAddress
2	(m)	Attribute:	RemoteDlcepAddress
DLL SERVICES:			
1	(m)	OpsService:	DL-IDE
2	(m)	OpsService:	DL-STA
2	(m)	OpsService:	DL-REQ

9.4.2.2 Attributes

LocalDlcepAddress
This attribute specifies the local DLCEP address and identifies the DLCEP. The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

RemoteDlcepAddress
This attribute specifies the remote DLCEP address and identifies the DLCEP.

9.4.2.3 DLL services

Refer to IEC 61158-3-13 for DLL service descriptions.

9.4.3 QUB-CL ARPM state machine

9.4.3.1 QUB-CL ARPM states

The QUB-CL ARPM state machine has only one state called "ACTIVE", see Figure 7.

Figure 7 – State transition diagram of QUB-CL ARPM

9.4.3.2 QUB-CL ARPM state table

Table 23 and Table 24 define the state machine of the QUB-CL ARPM.

Table 23 – QUB-CL ARPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	ACTIVE	Ident.req
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 5h — "Asyn1"
		requested-service-id := 1h — "ident-request"
		data := Ident.req)
		}
			ACTIVE

S2	ACTIVE	Status.req
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 5h — "Asyn1"
		requested-service-id := 2h — "status-request"
		data := Status.req)
		}
			ACTIVE

S3	ACTIVE	NMT-req-invite.req
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 5h — "Asyn1"
		requested-service-id := 2h — "NMT-req-invite"
		data := NMT-req-invite.req)
		}
			ACTIVE

S4	ACTIVE	Ident.rsp
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 6h — "Asyn2"
		service-id := 1h — "ident-response"
		data := Ident.rsp)
		}
			ACTIVE

S5	ACTIVE	Status.rsp
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 6h — "Asyn2"
		service-id := 2h — "status-response"
		data := Status.rsp)
		}
			ACTIVE

S6	ACTIVE	NMT-req-invite.rsp
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 6h — "Asyn2"
		service-id := 3h — "NMT-request"
		data := NMT-req-invite.rsp)
		}
			ACTIVE

NOTE Transactions S1 through S3 are executed by the MN only, transactions S4 through S6 are executed by CNs only.			

Table 24 – QUB-CL ARPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state
R1	ACTIVE	FAL-PDU_ind && requested-service-id := 1h — "ident-request" ⇒ Ident.ind	ACTIVE
R2	ACTIVE	FAL-PDU_ind && requested-service-id := 2h — "status-request" ⇒ Status.ind	ACTIVE
R3	ACTIVE	FAL-PDU_ind && requested-service-id := 3h — "NMT-req-invite" ⇒ NMT-req-invite.ind	ACTIVE
R4	ACTIVE	FAL-PDU_ind && service-id := 1h — "ident-response" ⇒ Ident.cnf	ACTIVE
R5	ACTIVE	FAL-PDU_ind && service-id := 2h — "status-response" ⇒ Status.cnf	ACTIVE
R6	ACTIVE	FAL-PDU_ind && service-id := 3h — "NMT-request" ⇒ NMT-req-invite.cnf	ACTIVE
NOTE Transactions R1 through R3 are executed by CNs only, transactions R4 through R6 are executed by the MN and may be executed by CNs depending on their configuration.			

9.4.3.3 Functions used by QUB-CL ARPM

The receipt of a FAL-PDU_ind primitive is always followed by its decoding to derive its relevant parameters for the state machine. Thus this implicit function is not listed separately.

Table 25 defines the other function used by this state machine.

Table 25 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
message-type service-id message-type data additional information		dlsdu	
Function			
Builds a FAL-PDU out of the parameters given as input variables.			

9.5 Queued user-triggered bi-directional connection-oriented with segmentation (QUB-COS) ARPM

9.5.1 Overview

The QUB-COS ARPM is divided in two sub-sections, so called layers:

- Sequence layer, QUB-COS (SeqL)
- Command layer, QUB-COS (CmdL)

The sequence layer provides the service of a reliable bidirectional connection that guarantees that no messages are lost or duplicated and that all messages arrive in the correct order. There shall be a sequence number for each sent frame, and an acknowledgement for the

sequence number of the opposite node, as well a connection state and a connection acknowledge.

The command layer has to decide whether a large block of data can be transferred in one frame (expedited transfer) or if it must be segmented in several frames (segmented transfer).

9.5.2 QUB-COS (CmdL) primitive definitions

9.5.2.1 Primitives exchanged between QUB-COS (CmdL) ARPM and user

Table 26 and Table 27 list the primitives exchanged between the ARPM and the user.

Table 26 – Primitives issued by user to QUB-COS (CmdL) ARPM

Primitive name	Source	Associated parameters	Functions
SDO-download.req	user	AREP invoke-ID command-ID segment-size data-size OD-identifier payload-data	Refer to service data definitions in IEC 61158-5-13
SDO-down-mult.req	user	AREP invoke-ID command-ID segment-size OD-identifier (n) payload-data (n)	Refer to service data definitions in IEC 61158-5-13
SDO-upload.req	user	AREP invoke-ID command-ID OD-identifier	Refer to service data definitions in IEC 61158-5-13
SDO-up-mult.req	user	AREP invoke-ID command-ID OD-identifier (n)	Refer to service data definitions in IEC 61158-5-13
SDO-abort.req	user	AREP invoke-ID error-info	Refer to service data definitions in IEC 61158-5-13
SDO-download.rsp	user	AREP invoke-ID error-info	Refer to service data definitions in IEC 61158-5-13
SDO-down-mult.rsp	user	AREP invoke-ID error-info (n)	Refer to service data definitions in IEC 61158-5-13
SDO-upload.rsp	user	AREP invoke-ID segment-size data-size payload-data error-info	Refer to service data definitions in IEC 61158-5-13
SDO-up-mult.rsp	user	AREP invoke-ID segment-size data-size payload-data / error-info (n)	Refer to service data definitions in IEC 61158-5-13

Table 27 – Primitives issued by QUB-COS (CmdL) ARPM to user

Primitive name	Source	Associated parameters	Functions
SDO-download.ind	ARPM	AREP invoke-ID command-ID segment-size data-size OD-identifier payload-data	Refer to service data definitions in IEC 61158-5-13
SDO-down-mult.ind	ARPM	AREP invoke-ID command-ID segment-size data-size OD-identifier (n) payload-data (n)	Refer to service data definitions in IEC 61158-5-13
SDO-upload.ind	ARPM	AREP invoke-ID command-ID OD-identifier	Refer to service data definitions in IEC 61158-5-13
SDO-up-mult.ind	ARPM	AREP invoke-ID command-ID OD-identifier (n)	Refer to service data definitions in IEC 61158-5-13
SDO-abort.ind	user	AREp invoke-ID error-info	Refer to service data definitions in IEC 61158-5-13
SDO-download.cnf	ARPM	AREP invoke-ID error-info	Refer to service data definitions in IEC 61158-5-13
SDO-down-mult.cnf	ARPM	AREP invoke-ID error-info (n)	Refer to service data definitions in IEC 61158-5-13
SDO-upload.cnf	ARPM	AREP invoke-ID segment-size data-size payload-data error-info	Refer to service data definitions in IEC 61158-5-13
SDO-up-mult.cnf	ARPM	AREP invoke-ID segment-size data-size payload-data / error-info (n)	Refer to service data definitions in IEC 61158-5-13

9.5.2.2 Parameters of primitives

The parameters of the primitives are described in IEC 61158-5-13.

9.5.3 QUB-COS (CmdL) ARPM state machine

9.5.3.1 QUB-COS (CmdL) ARPM states

The QUB-COS (CmdL) ARPM state machine has only one state called "ACTIVE", see Figure 8.

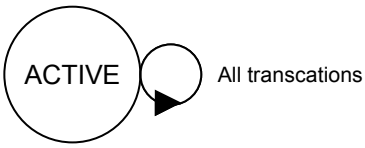


Figure 8 – State transition diagram of QUB-COS (CmdL) ARPM

9.5.3.2 QUB-COS (CmdL) ARPM state table

Table 28 and Table 29 define the state machine of the QUB-COS (CmdL) ARPM.

Table 28 – QUB-COS (CmdL) ARPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	ACTIVE	SDO-download.req || SDO-down-mult.req
		&& segment-size <= max-segment-size
		⇒
		response := 0
		abort := 0
		segmentation := 0
		data-size := "null"
		SEGMENT_req := BuildSegment (
		  header
		  segment-data)
			ACTIVE

S2	ACTIVE	SDO-download.req || SDO-down-mult.req
		&& segment-size > max-segment-size
		⇒
		response := 0
		abort := 0
		for i := 1 to (N := RoundUp(data-size, max-segment-size))
		  segmentation := 2
		  if (i = 1)
		  segmentation :=1
		  endif
		  if (i = N)
		  segmentation := 3
		  endif
		  SEGMENT_req := BuildSegment (
		  header
		  segment-data, i)
		endfor
		
		(see Notes)
			ACTIVE

S3	ACTIVE	SDO-download.rsp
		⇒
		response := 1
		abort := 0
		segmentation := 0
		data-size := "null"
		SEGMENT_req := BuildSegment (
		  header
		  segment-data := "null")
			ACTIVE

S4	ACTIVE	SDO-down-mult.rsp
		&& all data successfully written
		⇒
		response := 1
		abort := 0
		segmentation := 0
		data-size := "null"
		SEGMENT_req := BuildSegment (
		  header
		  segment-data := "null")
			ACTIVE

S5	ACTIVE	SDO-down-mult.rsp
		&& at least one data transfer failed
		&& segment-size <= max-segment-size
		⇒
		response := 1
		abort := 1
		segmentation := 0
		data-size := "null"
		SEGMENT_req := BuildSegment (
		  header
		  segment-data)
			ACTIVE

#	Current state	Event or condition ⇒ action	Next state

S6	ACTIVE	SDO-down-mult.rsp
		&& at least one data transfer failed
		&& segment-size > max-segment-size
		⇒
		response := 1
		abort := 1
		for i := 1 to (N := RoundUp(data-size, max-segment-size))
		segmentation := 2
		if (i = 1)
		segmentation :=1
		endif
		if (i = N)
		segmentation := 3
		endif
		SEGMENT_req := BuildSegment (
		header
		segment-data, i)
		endfor
		
		(see Notes)
			ACTIVE

S7	ACTIVE	SDO-upload.req ⇒ response :=0 abort := 0 segmentation := 0 data-size := "null" SEGMENT_req := BuildSegment (header segment-data := "null")	ACTIVE
S8	ACTIVE	SDO-upload.rsp && segment-size <= max-segment-size ⇒ response := 1 abort := 0 segmentation := 0 data-size := "null" SEGMENT _req := BuildSegment (header segment-data)	ACTIVE

S9	ACTIVE	SDO-upload.rsp
		&& segment-size > max-segment-size
		⇒
		response := 1
		abort := 0
		for i := 1 to (N := RoundUp(data-size, max-segment-size))
		segmentation := 2
		if (i = 1)
		segmentation :=1
		endif
		if (i = N)
		segmentation := 3
		endif
		SEGMENT_req := BuildSegment (
		header
		segment-data, i)
		endfor
		
		(see Notes)
			ACTIVE

S10	ACTIVE	SDO-up-mult.req && segment-size <= max-segment-size ⇒ response := 0 abort := 0 segmentation := 0 data-size := "null" SEGMENT_req := BuildSegment (header segment-data)	ACTIVE

#	Current state	Event or condition ⇒ action	Next state

S11	ACTIVE	SDO-up-mult.req
		&& segment-size > max-segment-size
		⇒
		response := 0
		abort := 0
		segmentation := 1
		for i := 1 to (N := RoundUp(data-size, max-segment-size))
		  segmentation := 2
		  if (i = 1)
		  segmentation :=1
		  endif
		  if (i = N)
		  segmentation := 3
		  endif
		  SEGMENT_req := BuildSegment (
		  header
		  segment-data, i)
		endfor
		
		(see Notes)
			ACTIVE

S12	ACTIVE	SDO-up-mult.rsp
		&& segment-size <= max-segment-size
		⇒
		response := 1
		if all data were successfully read
		  abort := 0
		else
		  abort := 1
		endif
		segmentation := 0
		data-size := "null"
		SEGMENT_req := BuildSegment (
		  header
		  segment-data)
			ACTIVE

S13	ACTIVE	SDO-up-mult.rsp
		&& segment-size > max-segment-size
		⇒
		response := 1
		if all data were successfully read
		  abort := 0
		else
		  abort := 1
		endif
		segmentation := 1
		for i := 1 to (N := RoundUp(data-size, max-segment-size))
		  segmentation := 2
		  if (i = 1)
		  segmentation :=1
		  endif
		  if (i = N)
		  segmentation := 3
		  endif
		  SEGMENT_req := BuildSegment (
		  header
		  segment-data, i)
		endfor
		
		(see Notes)
			ACTIVE

S14	ACTIVE	SDO-abort.req
		⇒
		abort := 1
		segmentation := 0
		data-size := "null"
		SEGMENT_req := BuildSegment (
		  header
		  segment-data := "error-info")
			ACTIVE

NOTE 1 When the length of the data exceeds the value of the "max-segment-size" parameter the QUB_COS (CmdL) protocol splits the payload data into N segment-data.
NOTE 2 For each segment-data, the function “BuildSegment” builds a Segment := Header with Command Layer parameters followed with segment-data without any gap.
NOTE 3 The segments reach the receiver AREP in the same order as they were created. This is guaranteed by the Sequence layer. Thus an additional numbering of the segments is not necessary.

Table 29 – QUB-COS (CmdL) ARPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state
R1	ACTIVE	SEGMENT_ind && response = 0 && abort = 0 && segmentation = 0 && (command-ID = 1h — "write-by-index" || command-ID = 3h — "write-all-by-index" ⇒ SDO-download.ind	ACTIVE

R2	ACTIVE	SEGMENT_ind
		&& response = 0
		&& abort = 0
		&& segmentation <> 0
		&& (command-ID = 1h — "write-by-index"
		|| command-ID = 3h — "write-all-by-index")
		&& AddSegment(segment) = "OK"
		⇒
		if (MoreFollows(segment) = “False”
		SDO-download.ind
		endif
		
		(see Note)
			ACTIVE

R3	ACTIVE	SEGMENT_ind && response = 1 && abort = 0 && (command-ID = 1h — "write-by-index" || command-ID = 3h — "write-all-by-index" ⇒ SDO-download.cnf	ACTIVE
R4	ACTIVE	SEGMENT_ind && response = 0 && abort = 0 && segmentation = 0 && command-ID = 31h — "write-multiple-by-index" ⇒ SDO-down-mult.ind	ACTIVE

R5	ACTIVE	SEGMENT_ind
		&& response = 0
		&& abort = 0
		&& segmentation <> 0
		&& command-ID = 31h — "write-multiple-by-index"
		&& AddSegment(segment) = "OK"
		⇒
		if (MoreFollows(segment) = “False”
		SDO-down-mult.ind
		endif
		
		(see Note)
			ACTIVE

R6	ACTIVE	SEGMENT_ind && response = 1 && abort = 0 || abort = 1 && segmentation = 0 && command-ID = 31h — "write-multiple-by-index" ⇒ SDO-down-mult.cnf	ACTIVE

R7	ACTIVE	SEGMENT_ind
		&& response = 1
		&& abort = 1
		&& segmentation <> 0
		&& command-ID = 31h — "write-multiple-by-index"
		&& AddSegment(segment) = "OK"
		⇒
		if (MoreFollows(segment) = “False”
		SDO-down-mult.cnf
		endif
		
		(see Note)
			ACTIVE

#	Current state	Event or condition ⇒ action	Next state
R8	ACTIVE	SEGMENT_ind && response = 0 && abort = 0 && (command-ID = 2h — "read-by-index" || command-ID = 4h — "read-all-by-index" ⇒ SDO-upload.ind	ACTIVE
R9	ACTIVE	SEGMENT_ind && response = 1 && abort = 0 && segmentation = 0 && (command-ID = 2h — "read-by-index" || command-ID = 4h — "read-all-by-index" ⇒ SDO-upload.cnf	ACTIVE

R10	ACTIVE	SEGMENT_ind
		&& response = 1
		&& abort = 0
		&& segmentation <> 0
		&& (command-ID = 2h — "read-by-index"
		|| command-ID = 4h — "read-all-by-index"
		&& AddSegment(segment) = "OK"
		⇒
		if (MoreFollows(segment) = “False”
		SDO-upload.cnf
		endif
		
		(see Note)
			ACTIVE

R11	ACTIVE	SEGMENT_ind && response = 0 && abort = 0 && segmentation = 0 && command-ID = 32h — "read-multiple-by-index" ⇒ SDO-up-mult.ind	ACTIVE

R12	ACTIVE	SEGMENT_ind
		&& response = 0
		&& abort = 0
		&& segmentation <> 0
		&& command-ID = 32h — "read-multiple-by-index"
		&& AddSegment(segment) = "OK"
		⇒
		if (MoreFollows(segment) = “False”
		SDO-up-mult.ind
		endif
		
		(see Note)
			ACTIVE

R13	ACTIVE	SEGMENT_ind && response = 1 && abort = 0 || abort = 1 && segmentation = 0 && command-ID = 32h — "read-multiple-by-index" ⇒ SDO-up-mult.cnf	ACTIVE

R14	ACTIVE	SEGMENT_ind
		&& response = 1
		&& abort = 0 || abort = 1
		&& segmentation <> 0
		&& command-ID = 32h — "read-multiple-by-index"
		&& AddSegment(segment) = "OK"
		⇒
		if (MoreFollows(segment) = “False”
		SDO-up-mult.cnf
		endif
		
		(see Note)
			ACTIVE

R15	ACTIVE	SEGMENT_ind && abort = 1 ⇒ SDO-abort.ind	ACTIVE

#	Current state	Event or condition ⇒ action	Next state
NOTE When the length of the data exceeds the value of the "max-segment-size" parameter the payload data are split into N segment-data. The segments are delivered in the order as they were created. Each segment contains a header with additional information. On the receiver end of the AR the function “AddSegment” removes this header (including "data-size" in the first segment) and appends the segment-data to the previous received segment-data. Once the N Segment-data are appended together without any gap, the function “GetintermediatePDU” gives the original OD entry identifier with their related payload data.			

9.5.3.3 Functions used by QUB-COS (CmdL) ARPM

The receipt of a SEGMENT_ind primitive is always followed by its decoding to derive its relevant parameters for the state machine. Thus this implicit function is not listed separately.

Table 30 through Table 34 define the other functions used by this state machine.

Table 30 – Function BuildSegment

Name	BuildSegment	Used in	ARPM
Input		Output	
header := AREP invoke-ID response abort segmentation command-ID segment-size segment-data := data-size (only for the first segment) data containing OD identifier(s) and related payload data, or error info (if applicable)		SEGMENT_req	
Function			
Builds a SEGMENT out of the parameters given as input variables. There is no additional segment number necessary as the correct order of segments is already guaranteed by the Sequence Layer already. This function adds the specified header to each segment. These headers are identical for all coherent segments with the exception that the parameter "segmentation" will have the value 1 for the first segment, 3 for the last and 2 for all segments in between. The parameter "data-size" will be provided only with the first segment.			

Table 31 – Function RoundUp

Name	BuildSegment	Used in	ARPM
Input		Output	
data-size max-segment-size		integer	
Function			
divides "data-size" by "max-segment-size" and rounds up the result to the next higher integer			

Table 32 – Function MoreFollows

Name	AddSegment	Used in	ARPM
Input		Output	
segmentation		Boolean	
Function			
This function inspects the "segmentation" parameter of the SEGMENT_ind header. If its value is 3, the function sets its output to "False"			

NOTE The following two functions make use of a persistent variable "IntermediatePDU", in which all segments of the current user data are stored by the receiver of this data.

Table 33 – Function AddSegment

Name	AddSegment	Used in	ARPM
Input		Output	
header := invoke-ID response abort segmentation command-ID segment-size data-size (first segment only) segment-data		Error code	
Function			
This function removes the header of the received SEGMENT_ind and appends the Segment_data to the previous received Segment_data. Once the N Segment_data are appended together without any gap, the function "GetintermediatePDU" gives the original OD entry identifier(s) with their related payload data.			

Table 34 – Function GetIntermediatePDU

Name	GetIntermediatePDU	Used in	ARPM
Input		Output	
(none)		OD entry identifier(s) with their related payload data	
Function			
This function returns the original data, which was received in multiple segments. After this function call the variable IntermediatePDU is reset and does not contain any segments.			

9.5.4 QUB-COS (SeqL) primitive definitions

9.5.4.1 Primitives exchanged between QUB-COS (SeqL) and QUB-COS (CmdL)

Table 35 shows the primitives issued by QUB-COS (CmdL) to QUB-COS (SeqL).

Table 35 – Primitives issued by QUB-COS (CmdL) to QUB-COS (SeqL)

Primitive names	Source	Associated parameters	Functions
SEGMENT_req	QUB-COS (CmdL)	AREP invoke-ID response abort segmentation command-ID segment-size data-size OD-identifier(s) data	This primitive is used to request the QUB-COS (SeqL) to transfer a data segment, It also carries information about the segmentation which will be needed at the destination AREP to reconstruct the complete message

Table 36 shows the primitives issued by QUB-COS (SeqL) to QUB-COS (CmdL).

Table 36 – Primitives issued by QUB-COS (SeqL) to QUB-COS (CmdL)

Primitive names	Source	Associated parameters	Functions
SEGMENT_ind	QUB-COS (SeqL)	AREP invoke-ID response abort segmentation command-ID segment-size data-size OD-identifier(s) data	This primitive is used to transmit a data segment from QUB-COS (SeqL) to QUB-COS (CmdL). In case of segmentation it contains the data segment itself and additional information about the segmentation from the remote AREP to reconstruct the complete message in the local QUB-COS (CmdL)

9.5.4.2 Parameters of primitives

The parameters used with the primitives exchanged between the QUB-COS (SeqL) and the QUB-COS (CmdL) are described in Table 37.

Table 37 – Parameters used with primitives exchanged between QUB-COS (SeqL) and QUB-COS (CmdL)

Parameter name	Description
AREP, invoke-ID, error-info	These parameters are as defined in IEC 61158-1.
response	This parameter indicates whether the SEGMENT contains a request or a response.
abort	This parameter indicates that the requested transfer could not be completed.
segmentation	This parameter indicates whether the SEGMENT is part of a segmented transfer or not.
command-ID	Specifies the command.
segment-size	In case of a segmented transfer this parameter contains the length of segment data.
data size	In case of a segmented transfer this parameter contains the total length of the transferred data block.
OD-identifier(s)	Identifies the OD entries to be affected.
data	Payload data to /from the OD entries.

9.5.5 DLL mapping of QUB-COS AREP Class

9.5.5.1 Formal model

This subclause describes the mapping of the QUB-COS AREP class to the Type 13 data link layer defined in IEC 61158-3-13 and IEC 61158-4-13. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the data link layer standard; rather, it defines how they are used by this AR class.

NOTE A means to configure and monitor the values of these attributes are not in the scope of this International Standard.

The DLL Mapping attributes and their permitted values and the DLL services used with the QUB-COS AREP class are defined in this subclause.

CLASS:			**Type 13 QUB-COS**
PARENT CLASS:			**Queued user-triggered bi directional-connection-oriented AREP**
ATTRIBUTES:			
1	(m)	KeyAttribute:	LocalDlcepAddress
2	(m)	Attribute:	RemoteDlcepAddress
3	(m)	Attribute:	RecSeqNr
4	(m)	Attribute:	RecCon
5	(m)	Attribute:	SndSeqNr
6	(m)	Attribute:	SndCon
DLL SERVICES:			
1	(m)	OpsService:	DL-SDO

9.5.5.2 Attributes

LocalDlcepAddress
This attribute specifies the local DLCEP address and identifies the DLCEP. The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

RemoteDlcepAddress
This attribute specifies the remote DLCEP address and identifies the DLCEP.

RecSeqNr
This attribute is a local buffer for the sequence number of the last correctly received frame.

RecConNr
This attribute is a local buffer for the acknowledge of connection code to the receiver.

SndSeqNr
This attribute is a local buffer for the sequence number of the frame.

SndCon
This attribute is a local buffer for the connection code of the sender.

9.5.5.3 DLL services

Refer to IEC 61158-3-13 for DLL service descriptions.

9.5.6 QUB-COS (SeqL) ARPM state machine

9.5.6.1 QUB-COS (SeqL) ARPM states

The defined states and their descriptions of the QUB-COS (SeqL) ARPM are shown in Table 38 and Figure 9.

Table 38 – QUB-COS (SeqL) ARPM states

State	Description
CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs. It only may send or receive FAL-PDUs with the parameter scon set to 1 to indicate the wish to initialize a connection.
REQ	The AREP acts as a client. It has sent a FAL-PDU with the parameter scon set to 1, and is waiting for a response from the remote server AREP.
RSP	The AREP acts as a server. It has received a FAL-PDU from the remote client AREP with the parameter scon set to 1, has returned a response Fal-PDU with the parameters scon := 1 and rcon := 1, and is waiting for a response from its user.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.

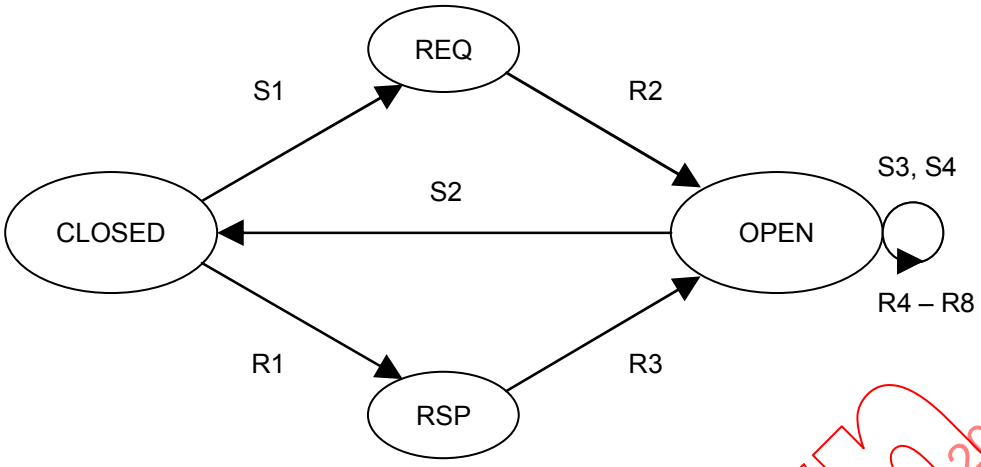


Figure 9 – State transition diagram of QUB-COS (SeqL) ARPM

9.5.6.2 QUB-COS (SeqL) ARPM state table

Table 39 and Table 40 define the state machine of the QUB-COS (SeqL) ARPM. In the comments of these tables "Node 1" indicates the initiator of a connection establishment process, "Node 2" indicates the addressed connection partner.