BS EN 61158-6-13:2014

BSI Standards Publication

Industrial communication networks — Fieldbus specifications

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

bsi.

...making excellence a habit.™

National foreword

This British Standard is the UK implementation of EN 61158-6-13:2014. It is identical to IEC 61158-6-13:2014. It supersedes BS EN 61158-6-13:2008 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee AMT/7, Industrial communications: process measurement and control, including fieldbus.

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

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

Published by BSI Standards Limited 2014

ISBN 978 0 580 79476 6
ICS 25.040.40; 35.100.70; 35.110

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 November 2014.

Amendments issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 61158-6-13

October 2014

ICS 25.040.40; 35.100.70; 35.110

Supersedes EN 61158-6-13:2008

English Version

Industrial communication networks - Fieldbus specifications - Part 6-13: Application layer protocol specification - Type 13 elements (IEC 61158-6-13:2014)

Réseaux de communication industriels - Spécifications des bus de terrain - Partie 6-13: Spécification du protocole de la couche application - Eléments de type 13 (CEI 61158-6-13:2014)

Industrielle Kommunikationsnetze - Feldbusse - Teil 6-13: Protokollspezifikation des Application Layer (Anwendungsschicht) - Typ 13-Elemente (IEC 61158-6-13:2014)

This European Standard was approved by CENELEC on 2014-09-23. 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 CEN-CENELEC Management Centre or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

CENELEC

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

CEN-CENELEC Management Centre: Avenue Marnix 17, B-1000 Brussels

Ref. No. EN 61158-6-13:2014 E

Foreword

The text of document 65C/764/FDIS, future edition 2 of IEC 61158-6-13, prepared by SC 65C "Industrial networks" of IEC/TC 65 "Industrial-process measurement, control and automation" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN 61158-6-13:2014.

The following dates are fixed:

• latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2015-06-23
• latest date by which the national standards conflicting with the document have to be withdrawn	(dow)	2017-09-23

This document supersedes EN 61158-6-13:2008.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC [and/or CEN] shall not be held responsible for identifying any or all such patent rights.

This document has been prepared under a mandate given to CENELEC by the European Commission and the European Free Trade Association.

Endorsement notice

The text of the International Standard IEC 61158-6-13:2014 was approved by CENELEC as a European Standard without any modification.

In the official version, for Bibliography, the following notes have to be added for the standards indicated:

IEC 61158-1	NOTE	Harmonized as EN 61158-1.
IEC 61158-6	NOTE	Harmonized as EN 61158-6 series.
IEC 61784-1	NOTE	Harmonized as EN 61784-1.
IEC 61784-2	NOTE	Harmonized as EN 61784-2.

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

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

NOTE 1 When an International Publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

NOTE 2 Up-to-date information on the latest versions of the European Standards listed in this annex is available here: www.cenelec.eu.

Publication	Year	Title	EN/HD	Year
IEC 61158-3-13	-	Industrial communication networks - Fieldbus specifications - Part 3-13: Data link layer service definition - Type 13 elements	EN 61158-3-13	-
IEC 61158-4-13	-	Industrial communication networks - Fieldbus specifications - Part 4-13: Data-link layer protocol specification - Type 13 elements	EN 61158-4-13	-
IEC 61158-5-13	-	Industrial communication networks - Fieldbus specifications - Part 5-13: Application layer service definition - Type 13 elements	EN 61158-5-13	-
ISO/IEC 7498	series	Information technology - Open Systems Interconnection - Basic reference model	-	-
ISO/IEC 7498-1	-	Information technology - Open Systems Interconnection - Basic reference model: The basic model	-	-
ISO/IEC 8802-3	-	Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications	-	-
ISO/IEC 8822	-	Information technology - Open Systems Interconnection - Presentation service definition	-	-
ISO/IEC 8824-1	-	Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation	-	-

Publication	Year	Title	EN/HD	Year
ISO/IEC 9545	-	Information technology - Open Systems Interconnection - Application layer structure	-	-
ISO/IEC 9899	-	Information technology - Programming languages - C	-	-
IEEE 754	-	IEEE Standard for Floating-Point Arithmetic	-	-

CONTENTS

INTRODUCTION ... 7
1 Scope ... 8
1.1 General ... 8
1.2 Specifications ... 8
1.3 Conformance ... 9
2 Normative references ... 9
3 Terms, definitions, symbols, abbreviations and conventions ... 9
3.1 ISO/IEC 7498-1 terms ... 10
3.2 ISO/IEC 8822 terms ... 10
3.3 ISO/IEC 9545 terms ... 10
3.4 ISO/IEC 8824-1 terms ... 10
3.5 Terms and definitions from IEC 61158-5-13 ... 11
3.6 Other terms and definitions ... 11
3.7 Abbreviations and symbols ... 11
4 FAL syntax description ... 12
4.1 General ... 12
4.2 FAL-AR PDU abstract syntax ... 12
4.3 Abstract syntax of Asyn1 pduBody ... 15
4.4 Abstract syntax of Asyn2 pduBody ... 16
5 Transfer syntax ... 23
5.1 Encoding of data types ... 23
6 FAL protocol state machines ... 27
7 AP context state machine ... 28
8 FAL service protocol machine ... 28
9 AR protocol machine ... 29
9.1 Buffered-network-scheduled bi-directional pre-established connection (BNB-PEC) ARPM ... 29
9.2 Buffered-network-scheduled uni-directional pre-established connection (BNU-PEC) ARPM ... 31
9.3 Queued user-triggered uni-directional (QUU) ARPM ... 33
9.4 Queued user-triggered bi-directional connectionless (QUB-CL) ARPM ... 36
9.5 Queued user-triggered bi-directional connection-oriented with segmentation (QUB-COS) ARPM ... 40
10 DLL mapping protocol machine ... 58
10.1 Primitive definitions ... 58
10.2 DMPM state machine ... 59
Annex A (normative) Constant value assignments ... 61
A.1 Values of abort-code ... 61
A.2 NMT-command-ID ... 62
A.3 Type 13 specific error-code constants ... 62
A.4 Node-list ... 64
Bibliography ... 65

Figure 1 – Encoding of Time of Day value ... 26

Figure 2 – Encoding of Time Difference value ... 27
Figure 3 – Primitives exchanged between protocol machines ... 28
Figure 4 – State transition diagram of BNB-PEC ARPM ... 30
Figure 5 – State transition diagram of BNU-PEC ARPM ... 32
Figure 6 – State transition diagram of QUU ARPM ... 35
Figure 7 – State transition diagram of QUB-CL ARPM ... 38
Figure 8 – State transition diagram of QUB-COS (CmdL) ARPM ... 43
Figure 9 – State transition diagram of QUB-COS (SeqL) ARPM ... 55
Figure 10 – State transition diagram of DMPM ... 59

Table 1 – Use of signaling-flags ... 14
Table 2 – Values of error-type ... 18
Table 3 – Transfer syntax for bit sequences ... 23
Table 4 – Transfer syntax for data type UNSIGNEDn ... 24
Table 5 – Transfer syntax for data type INTEGERn ... 25
Table 6 – Primitives issued by user to BNB-PEC ARPM ... 29
Table 7 – Primitives issued by BNB-PEC ARPM to user ... 29
Table 8 – BNB-PEC ARPM state table – sender transactions ... 30
Table 9 – BNB-PEC ARPM state table – receiver transactions ... 31
Table 10 – Function BuildFAL-PDU ... 31
Table 11 – Primitives issued by user to BNU-PEC ARPM ... 31
Table 12 – Primitives issued by BNU-PEC ARPM to user ... 31
Table 13 – BNU-PEC ARPM state table – sender transactions ... 33
Table 14 – BNU-PEC ARPM state table – receiver transactions ... 33
Table 15 – Function BuildFAL-PDU ... 33
Table 16 – Primitives issued by user to QUU ARPM ... 33
Table 17 – Primitives issued by QUU ARPM to user ... 34
Table 18 – QUU ARPM state table – sender transactions ... 35
Table 19 – QUU ARPM state table – receiver transactions ... 35
Table 20 – Function BuildFAL-PDU ... 36
Table 21 – Primitives issued by user to QUB-CL ARPM ... 36
Table 22 – Primitives issued by QUB-CL ARPM to user ... 37
Table 23 – QUB-CL ARPM state table – sender transactions ... 39
Table 24 – QUB-CL ARPM state table – receiver transactions ... 40
Table 25 – Function BuildFAL-PDU ... 40
Table 26 – Primitives issued by user to QUB-COS (CmdL) ARPM ... 41
Table 27 – Primitives issued by QUB-COS (CmdL) ARPM to user ... 42
Table 28 – QUB-COS (CmdL) ARPM state table – sender transactions ... 44
Table 29 – QUB-COS (CmdL) ARPM state table – receiver transactions ... 49
Table 30 – Function BuildSegment ... 51
Table 31 – Function RoundUp ... 51
Table 32 – Function MoreFollows ... 51

Table 33 – Function AddSegment 52
Table 34 – Function GetIntermediatePDU 52
Table 35 – Primitives issued by QUB-COS (CmdL) to QUB-COS (SeqL) 52
Table 36 – Primitives issued by QUB-COS (SeqL) to QUB-COS (CmdL) 53
Table 37 – Parameters used with primitives exchanged between QUB-COS (SeqL) and QUB-COS (CmdL) 53
Table 38 – QUB-COS (SeqL) ARPM states 54
Table 39 – QUB-COS (SeqL) ARPM state table – sender transactions 55
Table 40 – QUB-COS (SeqL) ARPM state table – receiver transactions 56
Table 41 – Function BuildFAL-PDU 58
Table 42 – Function IncrementCounter 58
Table 43 – Function AddToHistoryBuffer 58
Table 44 – Primitives issued by ARPM to DMPM 58
Table 45 – Primitives issued by DMPM to ARPM 58
Table 46 – Primitives issued by DMPM to data-link layer 59
Table 47 – Primitives issued by data-link layer to DMPM 59
Table 48 – DMPM state table – sender transactions 60
Table 49 – DMPM state table – receiver transactions 60
Table A.1 – Values of abort-code 61
Table A.2 – Values of NMTCommandID 62
Table A.3 – Type 13 specific error-code constants 63
Table A.4 – Node-list format 64

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 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 specifies 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 documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE All parts of the IEC 61158 series, as well as IEC 61784-1 and IEC 61784-2 are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this list of normative references.

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 7498-1, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*

ISO/IEC 8802-3, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*

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

ISO/IEC 8824-1, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*

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

ISO/IEC 9899, *Information technology – Programming languages – C*

IEEE 754, *IEEE Standard for Floating-Point Arithmetic*

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 8824-1 terms

For the purposes of this document, the following terms as defined in ISO/IEC 8824-1 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

3.7 Abbreviations and symbols

AE	Application entity
AL	Application layer
AP	Application process
APDU	Application protocol data unit
AR	Application relationship
AREP	Application relationship end point
ARPM	Application relationship protocol machine
ASnd	Asynchronous Send (Type 13 frame type)
BNB-PEC	Buffered network-scheduled bi-directional pre-established connection
BNU-PEC	Buffered network-scheduled uni-directional pre-established connection
CmdL	Command layer
CN	Controlled node

cnf	confirmation
DL-	(as a prefix) data-link-
DLCEP	Data-link connection end point
DLL	Data-link layer
DLME	Data-link-management entity
DLSAP	Data-link service access point
DLSDU	DL-service-data-unit
DMPM	DLL mapping protocol machine
DNS	Domain name service
FAL	Fieldbus application layer
ind	indication
IP	Internet protocol (see RFC 791)
MAC	Media access controll
MN	Managing node
NMT	Network management
OD	Object dictionary
PDO	Process data object
PDU	Process data unit
QUB-CL	Queued user-triggered bi-directional connectionless
QUB-COS	Queued user-triggered bi-directional connection-oriented with segmentation
QUU	Queued user-triggered uni-directional
req	request
rsp	response
SDO	Service data object
SeqL	Sequence layer
UDP	User datagram protocol

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
        fieldbus-version
        reserved8
        pduBody      CHOICE{
            [1h…5h] reserved
            [6h] Sync-request
            [7h…FFh] reserved
        }
}
```

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
        [6h] Sync-response
        [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...250, 253, 254)
```

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**	**IdentResponse**	**StatusResponse**	**SyncResponse**
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 Fieldbus-version

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

4.3 Abstract syntax of Asyn1 pduBody

4.3.1 Sync-request

4.3.1.1 Overview

```
Sync-request ::= SEQUENCE {
   synchronization-control   Bitstring     — see 4.3.1.2
   PRes-time                 Unsigned32    — time delay between end of the reception of the PRes from MN
                                             and start of sending the own time-triggered PRes in ns
   reserved                  Unsigned32
   sync-MN-delay             Unsigned32    — propagation delay between MN and CN in ns
   reserved                  Unsigned32
   fallback-timeout          Unsigned32    — SoC timeout for deactivating the time-triggered sending of PRes
                                             in state NMT_CS_PRE_OPERATIONAL_2 in ns
   destination-MAC-address   Unsigned32    — destination MAC address of the node the Sync-request is sent to
}
```

NOTE The above listed elements are sometimes summarized as follows:
"synchronization-control" through "destination-MAC-address" are summarized under the term "sync-control".

4.3.1.2 Synchronization-control

```
Synchronization-control ::= Bitstring {
    PRes-time-valid                (0)         — The parameter PRes-time is valid
    Reserved bit1                  (1)
    sync-MN-delay-valid            (2)         — The parameter sync-MN-delay is valid
    reserved bit2                  (3)
    fallback-timeout-valid         (4)         — The parameter fallback-timeout is valid
    reserved bit3                  (5)
    MAC-address-valid              (4)         — The parameter destination-MAC-address is valid
    reserved bit4 through bit26    (7)…(29)
    PRes-mode-reset                (30)        — Deactivate the time-triggered sending of PRes
    PRes-mode-set                  (31)        — Activate the time-triggered sending of PRes. This bit overules
                                                 bit30
}
```

4.4 Abstract syntax of Asyn2 pduBody

4.4.1 Ident-response

4.4.1.1 Overview

```
Ident-response ::= SEQUENCE {
    signaling-flags
    NMT-status
    reserved8
    fieldbus-version
    reserved8
    feature-flags                  BitString        — (see 4.4.1.2)
    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(48) OF Unsigned8
                                                    —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".

4.4.1.2 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
   WriteMultipleByIndex           (14)       — device supports WriteMultipleByIndex SDO service
   ReadMultipleByIndex            (15)       — device supports ReadMultipleByIndex SDO service
   reserved bit1 through bit2     (16)…(17)
   Time-triggered PRes            (18)       — device supports time-triggered sending of PRes
   reserved bit3 through bit16    (19)…(31)
}
```

4.4.2 Status-response

4.4.2.1 Overview

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

4.4.2.2 Static-error-bitfield

```
static-error-bit-field ::= SEQUENCE {
   error-register     BitString                              — (see 4.4.2.3)
   reserved8
   specific-errors    SEQUENCE SIZE (6) OF Unsigned8         — Device profile or vendor specific errors
}
```

4.4.2.3 Error-register

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

4.4.2.4 List-of-errors

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

4.4.2.5 ErrorEntry

```
ErrorEntry ::= SEQUENCE {
   error-type               Unsigned16          — (see 4.4.2.6)
   error-code               Unsigned16          — (see 4.4.2.6 and Clause A.3)
   time-stamp               Unsigned64
   additional-information   Unsigned64
}
```

4.4.2.6 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	Error-history entry
		1b	Status entry in Status-response frame (Bit 14 shall be set to 0b)
	14 (send)	0b	Error-history entry only
		1b	Additional to the error-history entry the entry shall also be entered in to the emergency queue of the error signaling
	13 .. 12 (mode)	0h	Not allowed in error-history entry. 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 Clause A.3)
		003h .. FFFh	error-code contains device profile specific errors

4.4.3 NMT-request

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

4.4.4 NMT-command

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

4.4.5 SDO

4.4.5.1 Overview

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

4.4.5.2 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
}
```

4.4.5.3 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
   }
}
```

4.4.5.4 Invoke ID

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

4.4.5.5 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
}
```

4.4.5.6 Command-ID

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

4.4.5.7 Segment-size

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

4.4.5.8 WriteByIndex services

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

WriteByIndex-ResponsePDU ::= NULL
```

4.4.5.9 ReadByIndex services

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

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

4.4.5.10 WriteAllByIndex services

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

WriteAllByIndex-ResponsePDU ::= NULL
```

4.4.5.11 ReadAllByIndex services

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

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

WriteByName-ResponsePDU ::= NULL
```

4.4.5.12 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                              — only present if write acces failed
   sub-index
   sub-abort
   sub-abort-code
   index                              — only present if write acces failed
   sub-index
   sub-abort
   sub-abort-code
   …                                  — (further write responses)
}
```

4.4.5.13 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)
}
```

4.4.5.14 AbortPDU

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

4.4.5.15 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.

4.4.5.16 Index

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

4.4.5.17 Sub-index

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

4.4.5.18 Payload-data

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

4.4.5.19 Offset

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

4.4.5.20 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

4.4.5.21 Sub-abort

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

4.4.5.22 Sub-abort-padding-length

sub-abort-padding-length ::= Unsigned8 — sub-abort (see 4.4.5.21) and padding-length (see 4.4.5.20) merged in one octet.

4.4.5.23 Sub-abort-code

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

4.4.6 Sync-response

4.4.6.1 Overview

```
Sync-response ::= SEQUENCE {
    synchronization-status  Bitstring    — see 4.4.6.2
    latency                 Unsigned32   — PRes latency in ns
    sync-node-number        Unsigned32   — node number received last inside SyncRequest/SyncResponse
    sync-delay              Unsigned32   — time difference between the end of receiving SyncRequest and the
                                           beginning of receiving the SyncResponse in ns
    PRes-time               Unsigned32   — time delay between reception of PRes from MN and time-triggered
                                           sending of the own PRes in ns
}
```

NOTE The above listed elements are sometimes summarized as follows:
"synchronization-status" through "destination-MAC-address" are summarized under the term "sync-status".

4.4.6.2 Synchronization-status

```
Synchronization-status ::= Bitstring {
    PRes-time-valid                (0)         — The parameter PRes-time is valid.
    reserved bit1 through bit30    (1)…(30)
    PRes-mode-status               (31)        — The time-triggered sending of PRes is active.
}
```

4.4.7 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 = $b_0 ... b_{n-1}$ 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.
	b_7 .. b_0	b_{15} .. b_8	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.

5.1.3 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.

5.1.4 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

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

is assigned the value

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

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 should 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.

5.1.5 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

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

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.

5.1.6 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 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 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

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

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.

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

$F = 2^{-23} \times (b_{22} \times 2^{22} + \dots + b_1 \times 2^1 + b_0 \times 2^0)$ 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.

5.1.7 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

5.1.8 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.

5.1.9 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

5.1.10 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 1984-01-01
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

5.1.11 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 Clause 6.

NOTE The state machines specified in Clause 6 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.

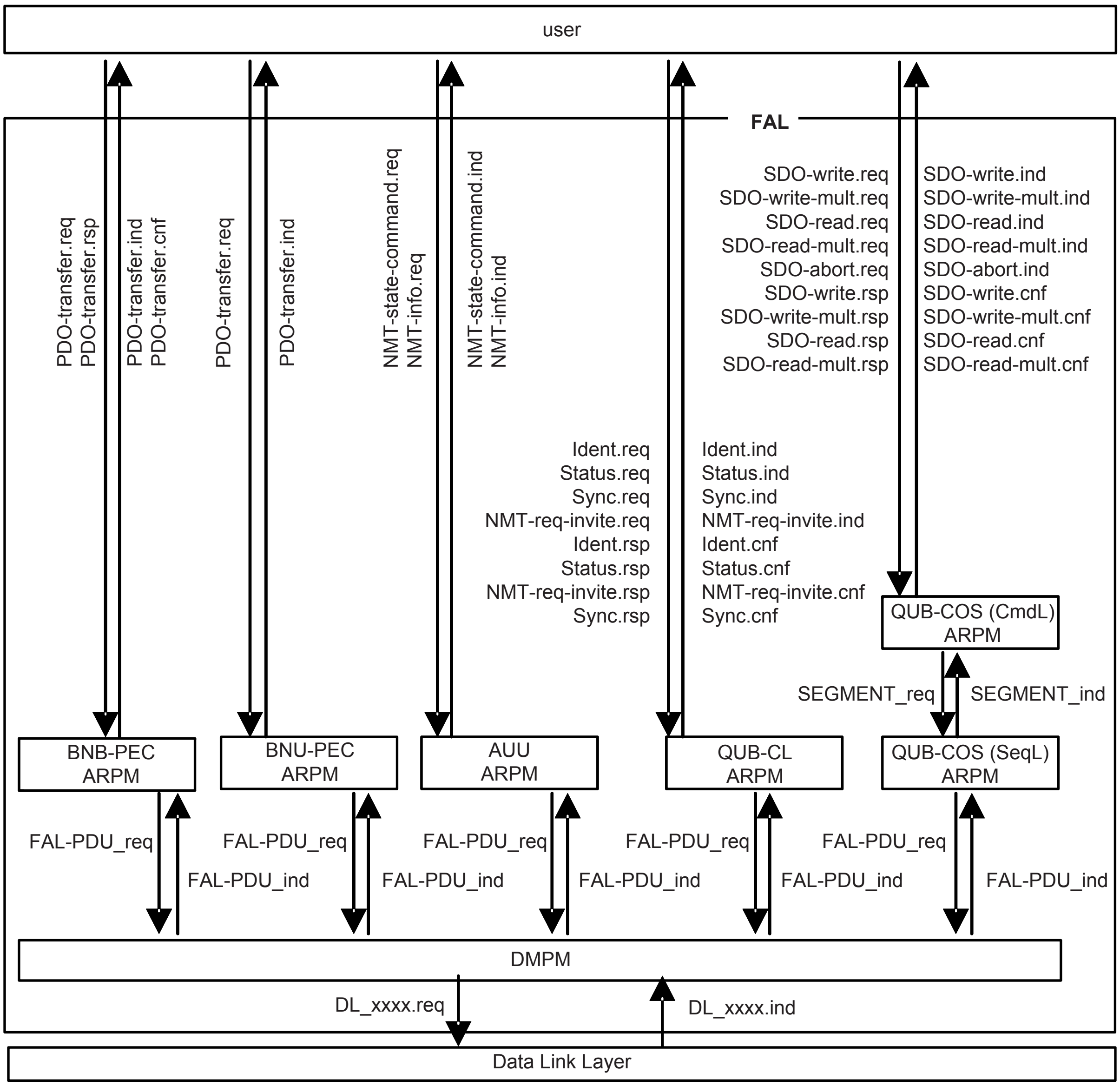

Figure 3 – Primitives exchanged between protocol machines

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

Subclause 9.1.2 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 is 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 9.1.2.

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.

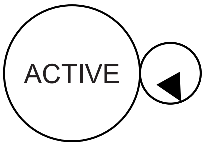

Figure 4 – State transition diagram of BNB-PEC ARPM

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 :="Isoc1"
		data := PDO-transfer.req)
		}
			ACTIVE

S2	ACTIVE	PDO-transfer.rsp
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type :="Isoc2"
		data := PDO-transfer.rsp)
		}
			ACTIVE

NOTE Transaction 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 = "Isoc1" ⇒ PDO-transfer.ind	ACTIVE
R2	ACTIVE	FAL-PDU_ind && message-type = "Isoc2" ⇒ 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

Subclause 9.2.2 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 is 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 9.2.2.

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.

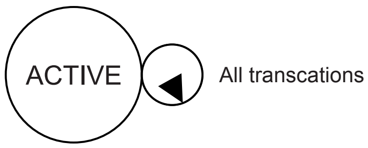

Figure 5 – State transition diagram of BNU-PEC ARPM

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 message-type data additional information		Output 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

Subclause 9.3.2 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 is 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 9.3.2.

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.

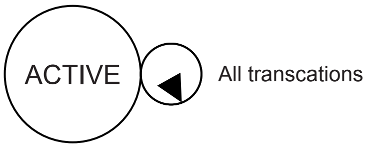

Figure 6 – State transition diagram of QUU ARPM

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 Clause 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 message-type service-id data additional information		Output 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 fieldbus-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
Sync.req	user	AREP sync-control	Refer to service data definitions in IEC 61158-5-13
Sync.rsp	user	AREP sync-status	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
Sync.ind	ARPM	AREP sync-control	Refer to service data definitions in IEC 61158-5-13
Ident.cnf	ARPM	AREP NMT-status fieldbus-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
Sync.cnf	ARPM	AREP sync-status	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

Subclause 9.4.2 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 is 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 9.4.2.

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.

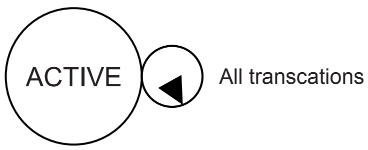

Figure 7 – State transition diagram of QUB-CL ARPM

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 := 3h — "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

S7	ACTIVE	Sync.req
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 5h — "Asyn1"
		requested-service-id := 6h — "sync-request"
		data := Sync.req)
		}
			ACTIVE

S8	ACTIVE	Sync.rsp
		⇒
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		message-type := 6h — "Asyn2"
		service-id := 6h — "sync-response"
		data := Sync.rsp)
		}
			ACTIVE

NOTE Transactions S1 through S3 and S7 are executed by the MN only, transactions S4 through S6 and S8 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
R7	ACTIVE	FAL-PDU_ind && requested-service-id := 6h — "sync-request" ⇒ Sync.ind	ACTIVE
R8	ACTIVE	FAL-PDU_ind && service-id := 6h — "sync-response" ⇒ Sync.cnf	ACTIVE
NOTE Transactions R1 through R3 and R7 are executed by CNs only, transactions R4 through R6 are executed by the MN and may be executed by CNs depending on their configuration, transaction R8 is executed by the MN and CNs.			

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 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-write.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-write-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-read.req	user	AREP invoke-ID command-ID OD-identifier	Refer to service data definitions in IEC 61158-5-13
SDO-read-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-write.rsp	user	AREP invoke-ID error-info	Refer to service data definitions in IEC 61158-5-13
SDO-write-mult.rsp	user	AREP invoke-ID error-info (n)	Refer to service data definitions in IEC 61158-5-13

Primitive name	Source	Associated parameters	Functions
SDO-read.rsp	user	AREP invoke-ID segment-size data-size payload-data error-info	Refer to service data definitions in IEC 61158-5-13
SDO-read-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-write.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-write-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-read.ind	ARPM	AREP invoke-ID command-ID OD-identifier	Refer to service data definitions in IEC 61158-5-13
SDO-read-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-write.cnf	ARPM	AREP invoke-ID error-info	Refer to service data definitions in IEC 61158-5-13

Primitive name	Source	Associated parameters	Functions
SDO-write-mult.cnf	ARPM	AREP invoke-ID error-info (n)	Refer to service data definitions in IEC 61158-5-13
SDO-read.cnf	ARPM	AREP invoke-ID segment-size data-size payload-data error-info	Refer to service data definitions in IEC 61158-5-13
SDO-read-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.

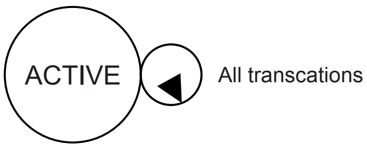

Figure 8 – State transition diagram of QUB-COS (CmdL) ARPM

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

#	Current state	Event or condition ⇒ action	Next state

S4	ACTIVE	SDO-write-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-write-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 := “null”)
			ACTIVE

S6	ACTIVE	SDO-write-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

#	Current state	Event or condition ⇒ action	Next state
S7	ACTIVE	SDO-read.req ⇒ response :=0 abort := 0 segmentation := 0 data-size := "null" SEGMENT_req := BuildSegment (header segment-data := "null")	ACTIVE
S8	ACTIVE	SDO-read.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-read.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

<table>
<tr><th>#</th><th>Current state</th><th>Event or condition
⇒ action</th><th>Next state</th></tr>
<tr><td>S10</td><td>ACTIVE</td><td>SDO-read-mult.req
&& segment-size <= max-segment-size
⇒
response := 0
abort := 0
segmentation := 0
data-size := "null"
SEGMENT_req := BuildSegment (
    header
    segment-data)</td><td>ACTIVE</td></tr>
<tr><td>S11</td><td>ACTIVE</td><td>SDO-read-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)</td><td>ACTIVE</td></tr>
<tr><td>S12</td><td>ACTIVE</td><td>SDO-read-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)</td><td>ACTIVE</td></tr>
</table>

#	Current state	Event or condition ⇒ action	Next state

S13	ACTIVE	SDO-read-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 ⇒ response := 0 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-write.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-write.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-write.cnf	ACTIVE
R4	ACTIVE	SEGMENT_ind && response = 0 && abort = 0 && segmentation = 0 && command-ID = 31h — "write-multiple-by-index" ⇒ SDO-write-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-write-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-write-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-write-mult.cnf
		endif
		
		(see Note)
			ACTIVE

<table>
<tr><th>#</th><th>Current state</th><th>Event or condition
⇒ action</th><th>Next state</th></tr>
<tr><td>R8</td><td>ACTIVE</td><td>SEGMENT_ind
&& response = 0
&& abort = 0
&& (command-ID = 2h — "read-by-index"
|| command-ID = 4h — "read-all-by-index"
⇒
SDO-read.ind</td><td>ACTIVE</td></tr>
<tr><td>R9</td><td>ACTIVE</td><td>SEGMENT_ind
&& response = 1
&& abort = 0
&& segmentation = 0
&& (command-ID = 2h — "read-by-index"
|| command-ID = 4h — "read-all-by-index"
⇒
SDO-read.cnf</td><td>ACTIVE</td></tr>
<tr><td>R10</td><td>ACTIVE</td><td>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-read.cnf
endif

(see Note)</td><td>ACTIVE</td></tr>
<tr><td>R11</td><td>ACTIVE</td><td>SEGMENT_ind
&& response = 0
&& abort = 0
&& segmentation = 0
&& command-ID = 32h — "read-multiple-by-index"
⇒
SDO-read-mult.ind</td><td>ACTIVE</td></tr>
<tr><td>R12</td><td>ACTIVE</td><td>SEGMENT_ind
&& response = 0
&& abort = 0
&& segmentation <> 0
&& command-ID = 32h — "read-multiple-by-index"
&& AddSegment(segment) = "OK"
⇒
if (MoreFollows(segment) = "False"
SDO-read-mult.ind
endif

(see Note)</td><td>ACTIVE</td></tr>
<tr><td>R13</td><td>ACTIVE</td><td>SEGMENT_ind
&& response = 1
&& abort = 0 || abort = 1
&& segmentation = 0
&& command-ID = 32h — "read-multiple-by-index"
⇒
SDO-read-mult.cnf</td><td>ACTIVE</td></tr>
<tr><td>R14</td><td>ACTIVE</td><td>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-read-mult.cnf
endif

(see Note)</td><td>ACTIVE</td></tr>
<tr><td>R15</td><td>ACTIVE</td><td>SEGMENT_ind
&& abort = 1
⇒
SDO-abort.ind</td><td>ACTIVE</td></tr>
</table>

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

Subclause 9.5.5 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 is 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 9.5.5.

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.

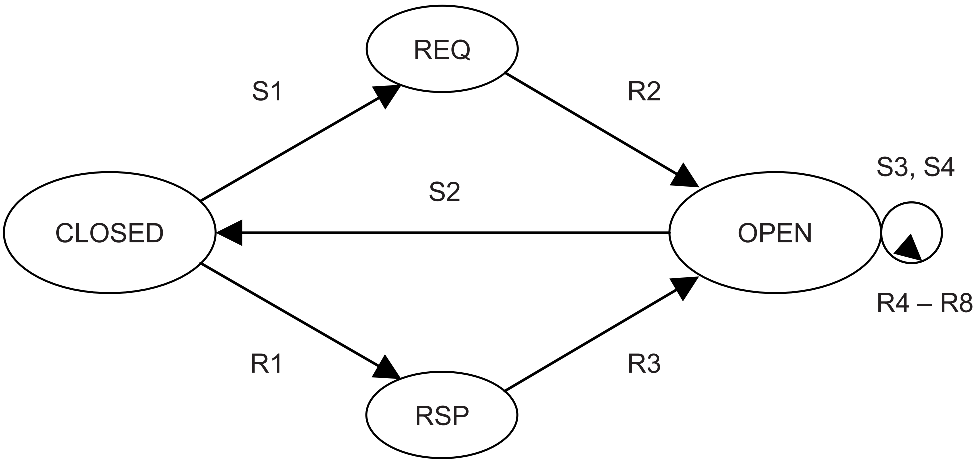

Figure 9 – State transition diagram of QUB-COS (SeqL) ARPM

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.

Table 39 – QUB-COS (SeqL) ARPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	CLOSED	SEGMENT_req — Node 1
		&& RecCon = 0, SndCon = 0
		&& RecSeqNr = ?, SndSeqNr = i
		⇒
		SndCon := 1
		
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		default-parameters,
		data := "null")
		}
			REQ

S2	OPEN	The connection is not needed any longer
		|| Delay of an expected response > SDO_SequLayerTimeout_U32
		⇒
		RecSeqNr := ?
		RecCon := 0
		SndSeqNr := ?
		SndCon := 0
		
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		default-parameters,
		data := "null")
		}
		
		NOTE The necessity for a connection is evaluated by means beyond the scope of this standard.
			CLOSED

S3	OPEN	SEGMENT_req — regular data transfer, sender
		&& RecCon = 2, SndCon = 2
		⇒
		IncrementCounter(SndSeqNr)
		
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		default-parameters,
		data := SEGMENT_req)
		}
		
		History(SndSecNr) = AddToHistoryBuffer()
			OPEN

#	Current state	Event or condition ⇒ action	Next state

S4	OPEN	SEGMENT_req — sender history full → ack request
		&& RecCon = 2, SndCon = 2
		&& SndSeqNr + 1 = D_SDO_SeqLayerTxHistorySize_U16
		⇒
		RecCon = 2
		RecSeqNr
		SndCon = 3
		IncrementCounter(SndSeqNr)
		
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		default-parameters,
		data := SEGMENT_req)
		}
			OPEN

Table 40 – QUB-COS (SeqL) ARPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state

R1	CLOSED	FAL-PDU_ind — Node 2
		&& rcon = 0, scon = 1
		&& ssnr = i
		&& SndSeqNr = j
		&& RecCon = 0, SndCon = 0
		⇒
		RecSeqNr := ssnr
		RecCon := 1
		SndCon := 1
		
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		default-parameters,
		data := "null")
		}
			RSP

R2	REQ	FAL-PDU_ind — Node 1
		&& rcon = 1, scon = 1
		&& ssnr = j, rsnr = i
		&& RecCon = 0, SndCon = 1
		⇒
		RecSeqNr := ssnr
		RecCon := 1
		SndSeqNr := rsnr
		SndCon := 2
		
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		default-parameters,
		data := "null")
		}
			OPEN

R3	RSP	FAL-PDU_ind — Node 2
		&& rcon = 1, scon = 2
		&& ssnr = i
		&& RecCon = 1, SndCon = 1
		⇒
		RecSeqNr := ssnr
		RecCon := 2
		SndSeqNr := rsnr
		SndCon := 2
		
		FAL-PDU_req {
		dlsdu := BuildFAL-PDU (
		default-parameters,
		data := "null")
		}
			OPEN

<table>
<tr><th>#</th><th>Current state</th><th>Event or condition
⇒ action</th><th>Next state</th></tr>
<tr><td>R4</td><td>OPEN</td><td>FAL-PDU_ind — regular data transfer, receiver
&& RecCon = 2, SndCon = 2
&& scon = 2, rcon = 2
&& ssnr = RecSeqNr +1
&& rsnr = SndSeqNr
⇒
RecSeqNr := ssnr
RecCon := 2
SndSeqNr := rsnr + 1
SndCon := 2

SEGMENT_ind</td><td>OPEN</td></tr>
<tr><td>R5</td><td>OPEN</td><td>FAL-PDU-ind — Loss of Frame detected
&& RecCon = 2, SndCon = 2
&& scon = 2, rcon = 2
&& ssnr > RecSeqNr +1
&& rsnr = SndSeqNr
⇒
RecCon := 3
RecSeqNr remains unchanged
SndCon := 2
SndSeqNr := rsnr

FAL-PDU_req {
dlsdu := BuildFAL-PDU (
default-parameters,
data := SEGMENT_req ! if present || "null")
}</td><td>OPEN</td></tr>
<tr><td>R6</td><td>OPEN</td><td>FAL-PDU_ind — data completion after loss of frame
&& RecCon = 2, SndCon = 2
&& scon= 2, rcon =3
&& rsnr < SndSeqNr
⇒
for i := rsnr+1 to SndSeqNr
FAL-PDU_req := History(i)
end for</td><td>OPEN</td></tr>
<tr><td>R7</td><td>OPEN</td><td>FAL-PDU-ind — Response to an ack request
&& scon = 3, rcon = 2
&& ssnr = RecSeqNr +1
&& rsnr = SndSeqNr
⇒
RecCon := 2
RecSeqNr := ssnr
SndSeqCon := 2
SndSeqNr := rsnr

FAL-PDU_req {
dlsdu := BuildFAL-PDU (
default-parameters,
data := SEGMENT_req ! if present || "null")
}</td><td>OPEN</td></tr>
<tr><td>R8</td><td>OPEN</td><td>FAL-PDU_ind — Duplication of frames detected
&& scon = 2
&& ssnr <= RecSeqNr.
⇒
drop FAL-PDU_ind, no further action</td><td>OPEN</td></tr>
</table>

9.5.6.3 Functions used by QUB-COS (SeqL) 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 41 through Table 43 define the other functions used by this state machine.

Table 41 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
default-parameters := message-type := 6h ("Asyn2") service-ID = 5h ("SDO") rsnr := RecSeqNr rcon := RecCon ssnr := SndSecNr scon := SndCon		dlsdu	
Function			
Builds a FAL-PDU out of the parameters given as input variables. The specified default values are valid for the use of this function in the QUB-COS (SeqL) ARPM only.			

Table 42 – Function IncrementCounter

Name	IncrementCounter	Used in	ARPM
Input		Output	
identifier			
Function			
This function increments the selected counter.			

Table 43 – Function AddToHistoryBuffer

Name	AddToHistoryBuffer	Used in	ARPM
Input		Output	
FAL-PDU_req SndSeqNr		History	
Function			
This function adds a sent FAL-PDU_req primitive to the history buffer of the state machine. The entry is referenced by the related SndSeqNr.			

10 DLL mapping protocol machine

10.1 Primitive definitions

10.1.1 Primitives exchanged between DMPM and ARPM

Table 44 and Table 45 list the primitives exchanged between DMPM and ARPM.

Table 44 – Primitives issued by ARPM to DMPM

Primitive name	Source	Associated parameters	Functions
FAL-PDU_req	ARPM	dlsdu	This primitive is used to request the DMPM to transfer an FAL-PDU, or to request an abort without transferring an FAL-PDU. It passes the FAL-PDU to the DMPM as a DLSDU.

Table 45 – Primitives issued by DMPM to ARPM

Primitive name	Source	Associated parameters	Functions
FAL-PDU_ind	DMPM	dlsdu	This primitive is used to pass an FAL-PDU received as a data link layer service data unit to a designated ARPM.

10.1.2 Parameters of ARPM / DMPM primitives

The "dlsdu" parameter contains the data of the application process and all relevant information for the state machine. The DMPM state machine is able to extract these informations.

10.1.3 Primitives exchanged between data-link layer and DMPM

Table 46 and Table 47 list the primitives exchanged between data-link layer and DMPM.

Table 46 – Primitives issued by DMPM to data-link layer

Primitive name	Source	Associated parameters
DL-PDO.req	DMPM	dl_dls_user_data
DL-CMD.req	DMPM	dl_ dls_user_data
DL-IDE.req	DMPM	dl_dls_user_data
DL-STA.req	DMPM	dl_dls_user_data
DL-REQ.req	DMPM	dl_dls_user_data
DL-SDO.req	DMPM	dl_dls_user_data

Table 47 – Primitives issued by data-link layer to DMPM

Primitive name	Source	Associated parameters
DL-PDO.ind	data-link layer	dl_dls_user_data
DL-CMD.ind	data-link layer	dl_ dls_user_data
DL-IDE.ind	data-link layer	dl_dls_user_data
DL-STA.ind	data-link layer	dl_dls_user_data
DL-REQ.ind	data-link layer	dl_dls_user_data
DL-SDO.ind	data-link layer	dl_dls_user_data

10.1.4 Parameters of DMPM / data-link layer primitives

The parameters used with the primitives exchanged between the DMPM and the data link layer are defined in the DLL Service definition (see IEC 61158-3-13). They are prefixed by “dl_” to indicate that they are used by the FAL.

10.2 DMPM state machine

10.2.1 DMPM states

The DMPM state machine has only one state called "ACTIVE", see Figure 10.

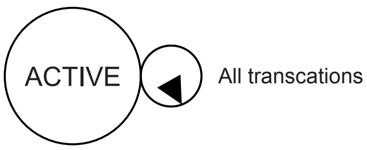

Figure 10 – State transition diagram of DMPM

10.2.2 DMPM state table

Table 48 and Table 49 define the state machine of the DMPM.

Table 48 – DMPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	ACTIVE	FAL-PDU_req
		&& message-type = 3h
		|| message-type = 4h
		⇒
		DL-PDO.req {
		dl_dls_user_data := dlsdu
		}
			ACTIVE

S2	ACTIVE	FAL-PDU_req
		&& service-ID = 4h
		⇒
		DL-CMD.req {
		dl_dls_user_data := dlsdu
		}
			ACTIVE

S3	ACTIVE	FAL-PDU_req
		&& requested-service-ID = 1h || service-ID = 1h
		⇒
		DL-IDE.req {
		dl_dls_user_data := dlsdu
		}
			ACTIVE

S4	ACTIVE	FAL-PDU_req
		&& requested-service-ID = 2h || service-ID = 2h
		⇒
		DL-STA.req {
		dl_dls_user_data := dlsdu
		}
			ACTIVE

S5	ACTIVE	FAL-PDU_req
		&& requested-service-ID = 3h || service-ID = 3h
		⇒
		DL-REQ.req {
		dl_dls_user_data := dlsdu
		}
			ACTIVE

S6	ACTIVE	FAL-PDU_req
		&& service-ID = 5h
		⇒
		DL-SDO.req {
		dl_dls_user_data := dlsdu
		}
			ACTIVE

Table 49 – DMPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state
R1	ACTIVE	DL-xxx.ind ⇒ FAL_PDU_ind	ACTIVE

10.2.3 Functions used by DMPM

The receipt of a DL_Put.ind or a DL_Data.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.

Annex A
(normative)

Constant value assignments

A.1 Values of abort-code

The valid values of the abort-code and their meaning are listed in Table A.1.

Table A.1 – Values of abort-code

Abort code	Description
0503 0000h	reserved
0504 0000h	SDO protocol timed out.
0504 0001h	Client/server command-ID not valid or unknown.
0504 0002h	Invalid block size.
0504 0003h	Invalid sequence number.
0504 0004h	reserved
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.
0604 0042h	The number and length of the objects to be mapped would exceed PDO length.
0604 0043h	General parameter incompatibility.
0604 0044h	Invalid heartbeat declaration
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 a file error).
0800 0024h	EDS, DCF or Concise DCF Data set empty.

A.2 NMT-command-ID

The valid values of the NMT-command-ID and their meaning are listed in Table A.2.

Table A.2 – Values of NMTCommandID

Name		ID Value
Plain NMT State Commands		20h .. 3Fh
	NMTStartNode	21h
	NMTStopNode	22h
	NMTEnterPreOperational2	23h
	NMTEnableReadyToOperate	24h
	NMTResetNode	28h
	NMTResetCommunication	29h
	NMTResetConfiguration	2Ah
	NMTSwReset	2Bh
Extended NMT State Commands		40h .. 5Fh
	NMTStartNodeEx	41h
	NMTStopNodeEx	42h
	NMTEnterPreOperational2Ex	43h
	NMTEnableReadyToOperateEx	44h
	NMTResetNodeEx	48h
	NMTResetCommunicationEx	49h
	NMTResetConfigurationEx	4Ah
	NMTSwResetEx	4Bh
NMT Info services		80h .. BFh
	NMTPublishConfiguredNodes	80h
	NMTPublishActiveNodes	90h
	NMTPublishPreOperational1	91h
	NMTPublishPreOperational2	92h
	NMTPublishReadyToOperate	93h
	NMTPublishOperational	94h
	NMTPublishStopped	95h
	NMTPublishNodeStates	96h
	NMTPublishEmergencyNew	A0h
	NMTPublishTime	B0h
NMTInvalidService		FFh

A.3 Type 13 specific error-code constants

The error-code constants specific for Type 13 are listed in Table A.3.

Table A.3 – Type 13 specific error-code constants

Name	Value	Description
	816xh	HW errors
E_DLL_BAD_PHYS_MODE	8161h	
E_DLL_COLLISION	8162h	
E_DLL_COLLISION_TH	8163h	
E_DLL_CRC_TH	8164h	
E_DLL_LOSS_OF_LINK	8165h	
E_DLL_MAC_BUFFER	8166h	
	82xxh	Protocol errors
E_DLL_ADDRESS_CONFLICT	8201h	
E_DLL_MULTIPLE_MN	8202h	
	821xh	Frame size errors
E_PDO_SHORT_RX	8210h	
E_PDO_MAP_VERS	8211h	
E_NMT_ASND_MTU_DIF	8212h	
E_NMT_ASND_MTU_LIM	8213h	
E_NMT_ASND_TX_LIM	8214h	
	823xh	Timing errors
E_NMT_CYCLE_LEN	8231h	
E_DLL_CYCLE_EXCEED	8232h	
E_DLL_CYCLE_EXCEED_TH	8233h	
E_NMT_IDLE_LIM	8234h	
E_DLL_JITTER_TH	8235h	
E_DLL_LATE_PRES_TH	8236h	
E_NMT_PREQ_CN	8237h	
E_NMT_PREQ_LIM	8238h	
E_NMT_PRES_CN	8239h	
E_NMT_PRES_RX_LIM	823Ah	
E_NMT_PRES_TX_LIM	823Bh	
	824xh	Frame errors
E_DLL_INVALID_FORMAT	8241h	
E_DLL_LOSS_PREQ_TH	8242h	
E_DLL_LOSS_PRES_TH	8243h	
E_DLL_LOSS_SOA_TH	8244h	
E_DLL_LOSS_SOC_TH	8245h	
E_DLL_LOSS_STATUSRES_TH	8246h	
	84xxh	BootUp errors
E_NMT_BA1	8410h	
E_NMT_BA1_NO_MN_SUPPORT	8411h	
E_NMT_BPO1	8420h	
E_NMT_BPO1_GET_IDENT	8421h	
E_NMT_BPO1_DEVICE_TYPE	8422h	
E_NMT_BPO1_VENDOR_ID	8423h	
E_NMT_BPO1_PRODUCT_CODE	8424h	
E_NMT_BPO1_REVISION_NO	8425h	
E_NMT_BPO1_SERIAL_NO	8426	
E_NMT_BPO1_CONFIGURATION	8428h	
E_NMT_BPO2	8430h	
E_NMT_BRO	8440h	
E_NMT_WRONG_STATE	8480h	

A.4 Node-list

The node-list assigns one bit for each Type 13 fieldbus node ID. The node ID assignment is given in Table A.4.

Table A.4 – Node-list format

Octet offset	Bit offset							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	7	6	5	4	3	2	1	-
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	63	62	61	60	59	58	57	56
8	71	70	69	68	67	66	65	64
9	79	78	77	76	75	74	73	72
10	87	86	85	84	83	82	81	80
11	95	94	93	92	91	90	89	88
12	103	102	101	100	99	98	97	96
13	111	110	109	108	107	106	105	104
14	119	118	117	116	115	114	113	112
15	127	126	125	124	123	122	121	120
16	135	135	133	132	131	130	129	128
17	143	142	141	140	139	138	137	136
18	151	150	149	148	147	146	145	144
19	159	158	157	156	155	154	153	152
20	167	166	165	164	163	162	161	160
21	175	174	173	172	171	170	169	168
22	183	182	181	180	179	178	177	176
23	191	190	189	188	187	186	185	184
24	199	198	197	196	195	194	193	192
25	207	206	205	204	203	202	201	200
26	215	214	213	212	211	210	209	208
27	223	222	221	220	219	218	217	216
28	231	230	229	228	227	226	225	224
29	239	238	237	236	235	234	233	232
30	247	246	245	244	243	242	241	240
31	-	254	253	252	251	250	249	248

Bibliography

IEC 61158-1, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-6 (all parts), *Industrial communication networks – Fieldbus specifications – Part 6: Application layer protocol specification*

IEC 61784-1, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

IEC 61784-2, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

EPSG DS 301 V1.2.0, *Ethernet POWERLINK Communication Profile Specification, Draft Standard Version 1.2.0, EPSG 2013,* available from http://www.ethernet-powerlink.com

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member**.

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email bsmusales@bsigroup.com.

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Copyright

Useful Contacts:

Customer Services
Tel: +44 845 086 9001
Email (orders): orders@bsigroup.com
Email (enquiries): cservices@bsigroup.com

Subscriptions
Tel: +44 845 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK