

INTERNATIONAL STANDARD

NORME INTERNATIONALE

**Industrial communication networks – Fieldbus specifications –
Part 4-2: Data-link layer protocol specification – Type 2 elements**

**Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 4-2: Spécification du protocole de la couche liaison de données –
Éléments de Type 2**



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2007 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester. If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'IEC ou du Comité national de l'IEC du pays du demandeur. Si vous avez des questions sur le copyright de l'IEC ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de l'IEC de votre pays de résidence.

IEC Central Office
3, rue de Varembe
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

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

About IEC publications

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

IEC Catalogue - webstore.iec.ch/catalogue

The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub

The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,...). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 14 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary

More than 55 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de l'IEC

La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications IEC

Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Catalogue IEC - webstore.iec.ch/catalogue

Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - www.iec.ch/searchpub

La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,...). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 14 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary

Plus de 55 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

INTERNATIONAL STANDARD

NORME INTERNATIONALE

**Industrial communication networks – Fieldbus specifications –
Part 4-2: Data-link layer protocol specification – Type 2 elements**

**Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 4-2: Spécification du protocole de la couche liaison de données –
Éléments de Type 2**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

PRICE CODE
CODE PRIX

XH

ICS 25.040.40; 35.100.20

ISBN 978-2-8322-1697-2

**Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.**

CONTENTS

FOREWORD.....	7
INTRODUCTION.....	9
1 Scope	10
1.1 General.....	10
1.2 Specifications	10
1.3 Procedures.....	10
1.4 Applicability.....	11
1.5 Conformance	11
2 Normative references	11
3 Terms, definitions, symbols and abbreviations.....	12
3.1 Reference model terms and definitions.....	12
3.2 Service convention terms and definitions.....	14
3.3 Common terms and definitions.....	15
3.4 Additional Type 2 definitions	16
3.5 Type 2 symbols and abbreviations	23
4 Overview of the DL-protocol.....	24
4.1 General.....	24
4.2 Services provided by the DL.....	26
4.3 Structure and definition of DL addresses.....	27
4.4 Services assumed from the PhL.....	29
4.5 Functional classes.....	31
5 General structure and encoding of PhPDUs and DLPDUs and related elements of procedure.....	32
5.1 Overview.....	32
5.2 Media access procedure.....	32
5.3 DLPDU structure and encoding.....	35
5.4 Lpacket components.....	39
5.5 DLPDU procedures.....	41
5.6 Summary of DLL support services and objects	42
6 Specific DLPDU structure, encoding and procedures	44
6.1 Modeling language	44
6.2 DLS user services	46
6.3 Generic tag Lpacket.....	52
6.4 Moderator Lpacket.....	53
6.5 Time distribution Lpacket.....	54
6.6 UCMM Lpacket.....	57
6.7 Keeper UCMM Lpacket	57
6.8 TUI Lpacket.....	58
6.9 Link parameters Lpacket and tMinus Lpacket	59
6.10 I'm-alive Lpacket	60
6.11 Ping Lpackets.....	62
6.12 WAMI Lpacket	64
6.13 Debug Lpacket	64
6.14 IP Lpacket.....	65
6.15 Ethernet Lpacket	65
7 Objects for station management.....	65

7.1	General	65
7.2	ControlNet object	66
7.3	Keeper object	76
7.4	Scheduling object	98
7.5	TCP/IP Interface object	109
7.6	Ethernet link object	118
7.7	DeviceNet object	124
7.8	Connection configuration object (CCO)	132
8	Other DLE elements of procedure	152
8.1	Network attachment monitor (NAM)	152
8.2	Calculating link parameters	159
9	Detailed specification of DL components	167
9.1	General	167
9.2	Access control machine (ACM)	167
9.3	TxLLC	184
9.4	RxLLC	188
9.5	Transmit machine (TxM)	191
9.6	Receive machine (RxM)	194
9.7	Serializer	200
9.8	Deserializer	201
9.9	DLL management	202
Annex A	(normative) Indicators and switches	205
A.1	Purpose	205
A.2	Indicators	205
A.3	Switches	216
Bibliography		218
INDEX		219
Figure 1	– Relationships of DLSAPs, DLSAP-addresses and group DL-addresses	15
Figure 2	– Data-link layer internal architecture	25
Figure 3	– Basic structure of a MAC ID address	27
Figure 4	– Basic structure of a generic tag address	27
Figure 5	– Basic structure of a fixed tag address	28
Figure 6	– M _s symbols and Manchester encoding at 5 MHz (informative)	29
Figure 7	– NUT structure	33
Figure 8	– Media access during scheduled time	33
Figure 9	– Media access during unscheduled time	34
Figure 10	– DLPDU format	35
Figure 11	– Aborting a DLPDU during transmission	39
Figure 12	– Lpacket format	39
Figure 13	– Generic tag Lpacket format	40
Figure 14	– Fixed tag Lpacket format	41
Figure 15	– Goodness parameter of TimeDist_Lpacket	55
Figure 16	– Example I'm alive processing algorithm	62
Figure 17	– Keeper CRC algorithm	82
Figure 18	– Keeper object power-up state diagram	94

Figure 19 – Keeper object operating state diagram	95
Figure 20 – Synchronized network change processing	98
Figure 21 – State transition diagram for TCP/IP Interface object	118
Figure 22 – Connection configuration object edit flowchart.....	152
Figure 23 – NAM state machine	153
Figure A.1 – Non redundant network status indicator labeling	209
Figure A.2 – Redundant network status indicator labeling	210
Table 1 – Data-link layer components.....	24
Table 2 – MAC ID addresses allocation	27
Table 3 – Fixed tag service definitions.....	28
Table 4 – Data encoding rules.....	29
Table 5 – M Data symbols.....	30
Table 6 – Truth table for ph_status_indication	31
Table 7 – FCS length, polynomials and constants.....	37
Table 8 – DLL support services and objects	43
Table 9 – Elementary data types	46
Table 10 – DLL events	51
Table 11 – Time distribution priority	56
Table 12 – Format of the TUI Lpacket	58
Table 13 – ControlNet object class attributes.....	67
Table 14 – ControlNet object instance attributes.....	67
Table 15 – TUI status flag bits.....	71
Table 16 – Channel state bits.....	72
Table 17 – ControlNet object common services	74
Table 18 – ControlNet object class specific services	75
Table 19 – Keeper object revision history	77
Table 20 – Keeper object class attributes	77
Table 21 – Keeper object instance attributes	77
Table 22 – Keeper operating state definitions	80
Table 23 – Port status flag bit definitions	80
Table 24 – TUI status flag bits.....	81
Table 25 – Keeper attributes	84
Table 26 – Memory requirements (in octets) for the Keeper attributes	84
Table 27 – Keeper object common services.....	85
Table 28 – Keeper object class specific services	86
Table 29 – Service error codes	87
Table 30 – Wire order format of the TUI Lpacket.....	91
Table 31 – Service error codes	92
Table 32 – Keeper object operating states.....	92
Table 33 – Keeper object state event matrix	96
Table 34 – Scheduling object class attributes	99
Table 35 – Scheduling object instance attributes	100

Table 36 – Scheduling object common services	100
Table 37 – Status error descriptions for Create	101
Table 38 – Status error descriptions for Delete and Kick_Timer	102
Table 39 – Scheduling object class specific services	102
Table 40 – Status error descriptions for Read	104
Table 41 – Status error descriptions for Conditional_Write	105
Table 42 – Status error descriptions for Forced_Write	105
Table 43 – Status error descriptions for Change_Start	106
Table 44 – Status error descriptions for Break_Connections	106
Table 45 – Status error descriptions for Change_Complete	107
Table 46 – Status error descriptions for Restart_Connections	108
Table 47 – TCP/IP Interface object class attributes	109
Table 48 – TCP/IP Interface object instance attributes	110
Table 49 – Status bits	112
Table 50 – Configuration capability bits	112
Table 51 – Configuration control bits	113
Table 52 – Example path	113
Table 53 – Interface configuration components	114
Table 54 – Alloc control values	115
Table 55 – TCP/IP Interface object common services	116
Table 56 – Get_Attribute_All reply format	117
Table 57 – Ethernet link object revision history	118
Table 58 – Ethernet link object class attributes	119
Table 59 – Ethernet link object instance attributes	119
Table 60 – Interface flags bits	121
Table 61 – Control bits	123
Table 62 – Ethernet Link object common services	123
Table 63 – Ethernet Link object class specific services	124
Table 64 – DeviceNet object revision history	124
Table 65 – DeviceNet object class attributes	125
Table 66 – DeviceNet object instance attributes	125
Table 67 – Bit rate attribute values	127
Table 68 – BOI attribute values	128
Table 69 – Diagnostic counters bit description	129
Table 70 – DeviceNet object common services	130
Table 71 – Reset service parameter	130
Table 72 – Reset service parameter values	131
Table 73 – DeviceNet object class specific services	131
Table 74 – Connection configuration object revision history	132
Table 75 – Connection configuration object class attributes	132
Table 76 – Format number values	134
Table 77 – Connection configuration object instance attributes	135
Table 78 – Originator connection status values	137

Table 79 – Target connection status values	138
Table 80 – Connection flags	138
Table 81 – I/O mapping formats	140
Table 82 – Connection configuration object common services	141
Table 83 – Get_Attribute_All error codes	141
Table 84 – Get_Attribute_All response	142
Table 85 – Set_Attribute_All error codes	143
Table 86 – Set_Attribute_All request	144
Table 87 – Create request parameters	145
Table 88 – Create error codes	146
Table 89 – Delete error codes	146
Table 90 – Restore error codes	146
Table 91 – Connection configuration object class specific services	147
Table 92 – Kick_Timer error codes	147
Table 93 – Open_Connection error codes	148
Table 94 – Close_Connection error codes	148
Table 95 – Stop_Connection error codes	148
Table 96 – Change_Start error codes	149
Table 97 – Get_Status service parameter	149
Table 98 – Get_Status service response	149
Table 99 – Get_Status service error codes	150
Table 100 – Change_Complete service parameter	150
Table 101 – Change_Complete service error codes	150
Table 102 – Audit_Changes service parameter	151
Table 103 – Audit_Changes service error codes	151
Table 104 – NAM states	153
Table 105 – Default link parameters	154
Table 106 – PhL timing characteristics	160
Table A.1 – Module status indicator	206
Table A.2 – Network status indicators	207
Table A.3 – Network status indicator	211
Table A.4 – Network status indicator	213
Table A.5 – Combined module/network status indicator	214
Table A.6 – I/O status indicator	215
Table A.7 – Bit rate switch encoding	217

INTERNATIONAL ELECTROTECHNICAL COMMISSION

**INDUSTRIAL COMMUNICATION NETWORKS –
FIELDBUS SPECIFICATIONS –****Part 4-2: Data-link layer protocol specification – Type 2 elements**

FOREWORD

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

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

IEC draws attention to the fact that it is claimed that compliance with this standard may involve the use of patents as follows, where the [xx] notation indicates the holder of the patent right:

Type 2 and possibly other types:

US 5,400,331	[RA]	Communication network interface with screeners for incoming messages
US 5,471,461	[RA]	Digital communication network with a moderator station election process
US 5,491,531	[RA]	Media access controller with a shared class message delivery capability
US 5,493,571	[RA]	Apparatus and method for digital communications with improved delimiter detection
US 5,537,549	[RA]	Communication network with time coordinated station activity by time slot and periodic interval number
US 5,553,095	[RA]	Method and apparatus for exchanging different classes of data during different time intervals

IEC takes no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holders of these patent rights are registered with IEC. Information may be obtained from

[RA]: Rockwell Automation, Inc.
1201 S. Second Street
Milwaukee, WI 53204
USA
Attention: Intellectual Property Dept.

Attention is drawn to the possibility that some of the elements of this standard may be the subject of patent rights other than those identified above. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61158-4-2 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This bilingual version (2014-07) corresponds to the English version, published in 2007-12.

This first edition and its companion parts of the IEC 61158-4 subseries cancel and replace IEC 61158-4:2003. This edition of this part constitutes a minor revision. This part and its companion Type 2 parts also cancel and replace IEC PAS 62413, published in 2005.

This edition of IEC 61158-4 includes the following significant changes from the previous edition:

- a) deletion of the former Type 6 fieldbus, and the placeholder for a Type 5 fieldbus data link layer, for lack of market relevance;
- b) addition of new types of fieldbuses,
- c) division of this part into multiple parts numbered -4-1, -4-2, ..., -4-19.

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

FDIS	Report on voting
65C/474/FDIS	65C/485/RVD

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

The French version of this standard has not been voted upon.

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

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

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

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

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

INTRODUCTION

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

The data-link protocol provides the data-link service by making use of the services available from the physical 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 data-link entities (DLEs) 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:

- a) as a guide for implementors and designers;
- b) for use in the testing and procurement of equipment;
- c) as part of an agreement for the admittance of systems into the open systems environment;
- d) 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 4-2: Data-link layer protocol specification – Type 2 elements

1 Scope

1.1 General

The data-link layer provides basic time-critical messaging communications between devices in an automation environment.

This protocol provides communication opportunities to all participating data-link entities, sequentially and in a cyclic synchronous manner. Foreground scheduled access is available for time-critical activities together with background unscheduled access for less critical activities.

Deterministic and synchronized transfers can be provided at cyclic intervals up to 1 ms and device separations of 25 km. This performance is adjustable dynamically and on-line by re-configuring the parameters of the local link whilst normal operation continues. By similar means, DL connections and new devices may be added or removed during normal operation.

This protocol provides means to maintain clock synchronization across an extended link with a precision better than 10 μ s.

This protocol optimizes each access opportunity by concatenating multiple DLSDUs and associated DLPCI into a single DLPDU, thereby improving data transfer efficiency for data-link entities that actively source multiple streams of data.

The maximum system size is an unlimited number of links of 99 nodes, each with 255 DLSAP-addresses. Each link has a maximum of 2^{24} related peer and publisher DLCEPs.

1.2 Specifications

This standard specifies

- a) procedures for the timely transfer of data and control information from one data-link user entity to a peer user entity, and among the data-link entities forming the distributed data-link service provider;
- b) the structure of the fieldbus DLPDUs used for the transfer of data and control information by the protocol of this standard, and their representation as physical interface data units.

1.3 Procedures

The procedures are defined in terms of

- a) the interactions between peer DL-entities (DLEs) through the exchange of fieldbus DLPDUs;
- b) the interactions between a DL-service (DLS) provider and a DLS-user in the same system through the exchange of DLS primitives;
- c) the interactions between a DLS-provider and a Ph-service provider in the same system through the exchange of Ph-service primitives.

1.4 Applicability

These procedures are applicable to instances of communication between systems which support time-critical communications services within the data-link layer of the OSI or fieldbus reference models, and which require the ability to interconnect in an open systems interconnection environment.

Profiles provide a simple multi-attribute means of summarizing an implementation's capabilities, and thus its applicability to various time-critical communications needs.

1.5 Conformance

This standard also specifies conformance requirements for systems implementing these procedures. This standard does not contain tests to demonstrate compliance with such requirements.

2 Normative references

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

IEC 61131-3, *Programmable controllers – Part 3: Programming languages*

IEC 61158-2 (Ed.4.0), *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 61158-3-2, *Industrial communication networks – Fieldbus specifications – Part 3-2: Data-link layer service definition – Type 2 elements*

IEC 61158-5-2, *Industrial communication networks – Fieldbus specifications – Part 5-2: Application layer service definition – Type 2 elements*

IEC 61158-6-2, *Industrial communication networks – Fieldbus specifications – Part 6-2: Application layer protocol specification – Type 2 elements*

IEC 61784-3-2, *Industrial communication networks – Profiles – Part 3-2: Functional safety fieldbuses – Additional specifications for CPF 2*

IEC 62026-3¹, *Low-voltage switchgear and controlgear – Controller-device interfaces (CDIs) – Part 3: DeviceNet*

ISO/IEC 3309, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures – Frame structure*

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

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing*

ISO/IEC 8802-3, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3:*

¹ To be published.

Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications

ISO/IEC 10731, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

ISO 11898:1993², *Road vehicles – Interchange of digital information – Controller area network (CAN) for high-speed communication*

3 Terms, definitions, symbols and abbreviations

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

3.1 Reference model terms and definitions

This standard is based in part on the concepts developed in ISO/IEC 7498-1 and ISO/IEC 7498-3, and makes use of the following terms defined therein.

3.1.1	called-DL-address	[ISO/IEC 7498-3]
3.1.2	calling-DL-address	[ISO/IEC 7498-3]
3.1.3	centralized multi-end-point-connection	[ISO/IEC 7498-1]
3.1.4	correspondent (N)-entities correspondent DL-entities (N=2) correspondent Ph-entities (N=1)	[ISO/IEC 7498-1]
3.1.5	demultiplexing	[ISO/IEC 7498-1]
3.1.6	DL-address	[ISO/IEC 7498-3]
3.1.7	DL-address-mapping	[ISO/IEC 7498-1]
3.1.8	DL-connection	[ISO/IEC 7498-1]
3.1.9	DL-connection-end-point	[ISO/IEC 7498-1]
3.1.10	DL-connection-end-point-identifier	[ISO/IEC 7498-1]
3.1.11	DL-connection-mode transmission	[ISO/IEC 7498-1]
3.1.12	DL-connectionless-mode transmission	[ISO/IEC 7498-1]
3.1.13	DL-data-sink	[ISO/IEC 7498-1]
3.1.14	DL-data-source	[ISO/IEC 7498-1]
3.1.15	DL-duplex-transmission	[ISO/IEC 7498-1]
3.1.16	DL-facility	[ISO/IEC 7498-1]
3.1.17	DL-local-view	[ISO/IEC 7498-3]
3.1.18	DL-name	[ISO/IEC 7498-3]
3.1.19	DL-protocol	[ISO/IEC 7498-1]

² A newer edition of this standard has been published, but only the cited edition applies.

3.1.20	DL-protocol-connection-identifier	[ISO/IEC 7498-1]
3.1.21	DL-protocol-control-information	[ISO/IEC 7498-1]
3.1.22	DL-protocol-data-unit	[ISO/IEC 7498-1]
3.1.23	DL-protocol-version-identifier	[ISO/IEC 7498-1]
3.1.24	DL-relay	[ISO/IEC 7498-1]
3.1.25	DL-service-connection-identifier	[ISO/IEC 7498-1]
3.1.26	DL-service-data-unit	[ISO/IEC 7498-1]
3.1.27	DL-simplex-transmission	[ISO/IEC 7498-1]
3.1.28	DL-subsystem	[ISO/IEC 7498-1]
3.1.29	DL-user-data	[ISO/IEC 7498-1]
3.1.30	flow control	[ISO/IEC 7498-1]
3.1.31	layer-management	[ISO/IEC 7498-1]
3.1.32	multiplexing	[ISO/IEC 7498-3]
3.1.33	naming-(addressing)-authority	[ISO/IEC 7498-3]
3.1.34	naming-(addressing)-domain	[ISO/IEC 7498-3]
3.1.35	naming-(addressing)-subdomain	[ISO/IEC 7498-3]
3.1.36	(N)-entity DL-entity Ph-entity	[ISO/IEC 7498-1]
3.1.37	(N)-interface-data-unit DL-service-data-unit (N=2) Ph-interface-data-unit (N=1)	[ISO/IEC 7498-1]
3.1.38	(N)-layer DL-layer (N=2) Ph-layer (N=1)	[ISO/IEC 7498-1]
3.1.39	(N)-service DL-service (N=2) Ph-service (N=1)	[ISO/IEC 7498-1]
3.1.40	(N)-service-access-point DL-service-access-point (N=2) Ph-service-access-point (N=1)	[ISO/IEC 7498-1]
3.1.41	(N)-service-access-point-address DL-service-access-point-address (N=2) Ph-service-access-point-address (N=1)	[ISO/IEC 7498-1]
3.1.42	peer-entities	[ISO/IEC 7498-1]
3.1.43	Ph-interface-control-information	[ISO/IEC 7498-1]
3.1.44	Ph-interface-data	[ISO/IEC 7498-1]
3.1.45	primitive name	[ISO/IEC 7498-3]

3.1.46	reassembling	[ISO/IEC 7498-1]
3.1.47	recombining	[ISO/IEC 7498-1]
3.1.48	reset	[ISO/IEC 7498-1]
3.1.49	responding-DL-address	[ISO/IEC 7498-3]
3.1.50	routing	[ISO/IEC 7498-1]
3.1.51	segmenting	[ISO/IEC 7498-1]
3.1.52	sequencing	[ISO/IEC 7498-1]
3.1.53	splitting	[ISO/IEC 7498-1]
3.1.54	synonymous name	[ISO/IEC 7498-3]
3.1.55	systems-management	[ISO/IEC 7498-1]

3.2 Service convention terms and definitions

This standard also makes use of the following terms defined in ISO/IEC 10731 as they apply to the data-link layer:

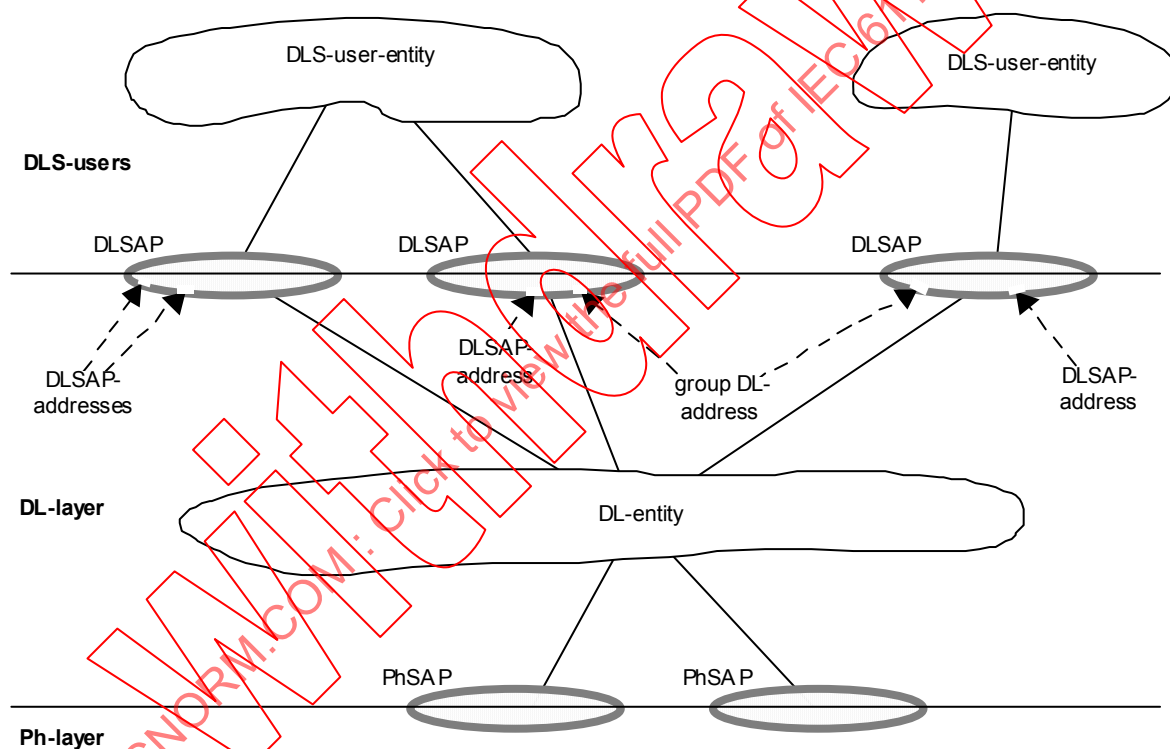
3.2.1	acceptor
3.2.2	asymmetrical service
3.2.3	confirm (primitive); requestor.deliver (primitive)
3.2.4	deliver (primitive)
3.2.5	DL-confirmed-facility
3.2.6	DL-facility
3.2.7	DL-local-view
3.2.8	DL-mandatory-facility
3.2.9	DL-non-confirmed-facility
3.2.10	DL-provider-initiated-facility
3.2.11	DL-provider-optional-facility
3.2.12	DL-service-primitive; primitive
3.2.13	DL-service-provider
3.2.14	DL-service-user
3.2.15	DL-user-optional-facility
3.2.16	indication (primitive) acceptor.deliver (primitive)
3.2.17	multi-peer
3.2.18	request (primitive); requestor.submit (primitive)

3.2.19 requestor**3.2.20 response (primitive);
acceptor.submit (primitive)****3.2.21 submit (primitive)****3.2.22 symmetrical service****3.3 Common terms and definitions**

NOTE Many definitions are common to more than one protocol Type; they are not necessarily used by all protocol Types.

3.3.1**DL-segment, link, local link**

single DL-subnetwork in which any of the connected DLEs may communicate directly, without any intervening DL-relaying, whenever all of those DLEs that are participating in an instance of communication are simultaneously attentive to the DL-subnetwork during the period(s) of attempted communication



NOTE 1 DLSAPs and PhSAPs are depicted as ovals spanning the boundary between two adjacent layers.

NOTE 2 DL-addresses are depicted as designating small gaps (points of access) in the DLL portion of a DLSAP.

NOTE 3 A single DL-entity may have multiple DLSAP-addresses and group DL-addresses associated with a single DLSAP.

Figure 1 – Relationships of DLSAPs, DLSAP-addresses and group DL-addresses

3.3.2**DLSAP**

distinctive point at which DL-services are provided by a single DL-entity to a single higher-layer entity.

NOTE This definition, derived from ISO/IEC 7498-1, is repeated here to facilitate understanding of the critical distinction between DLSAPs and their DL-addresses. (See Figure 1.)

3.3.3

DL(SAP)-address

either an individual DLSAP-address, designating a single DLSAP of a single DLS-user, or a group DL-address potentially designating multiple DLSAPs, each of a single DLS-user.

NOTE This terminology is chosen because ISO/IEC 7498-3 does not permit the use of the term DLSAP-address to designate more than a single DLSAP at a single DLS-user.

3.3.4

(individual) DLSAP-address

DL-address that designates only one DLSAP within the extended link

NOTE A single DL-entity may have multiple DLSAP-addresses associated with a single DLSAP.

3.3.5

extended link

DL-subnetwork, consisting of the maximal set of links interconnected by DL-relays, sharing a single DL-name (DL-address) space, in which any of the connected DL-entities may communicate, one with another, either directly or with the assistance of one or more of those intervening DL-relay entities

NOTE An extended link may be composed of just a single link.

3.3.6

frame

denigrated synonym for DLPDU

3.3.7

group DL-address

DL-address that potentially designates more than one DLSAP within the extended link. A single DL-entity may have multiple group DL-addresses associated with a single DLSAP. A single DL-entity also may have a single group DL-address associated with more than one DLSAP

3.3.8

node

single DL-entity as it appears on one local link

3.3.9

receiving DLS-user

DL-service user that acts as a recipient of DL-user-data

NOTE A DL-service user can be concurrently both a sending and receiving DLS-user.

3.3.10

sending DLS-user

DL-service user that acts as a source of DL-user-data

3.4 Additional Type 2 definitions

3.4.1

actual packet interval (API)

measure of how frequently a specific connection produces its Lpacket data

3.4.2

allocate

take a resource from a common area and assign that resource for the exclusive use of a specific entity

3.4.3

application

function or data structure for which data is consumed or produced

3.4.4**application objects**

multiple object classes that manage and provide the run time exchange of messages across the network and within the device

3.4.5**attribute**

description of an externally visible characteristic or feature of an object

NOTE The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behavior of an object. Attributes are divided into class attributes and instance attributes.

3.4.6**behavior**

indication of how the object responds to particular events. Its description includes the relationship between attribute values and services

3.4.7**bit**

unit of information consisting of a 1 or a 0. This is the smallest data unit that can be transmitted

3.4.8**blanking or blanking time**

length of time required after transmitting before the node is allowed to receive

3.4.9**client**

- 1) An object which uses the services of another (server) object to perform a task
- 2) An initiator of a message to which a server reacts

3.4.10**communication objects**

components that manage and provide run time exchange of messages across the network such as the ControlNet object, the Unconnected Message Manager (UCMM)

3.4.11**connection**

logical binding between two application objects within the same or different devices

3.4.12**connection ID (CID)**

identifier assigned to a transmission, associated with a particular connection between producers and consumers, that identifies a specific piece of application information

NOTE Within the data link this is a generic tag.

3.4.13**cyclic redundancy check (CRC)**

residual value computed from an array of data and used as a representative signature for the array

3.4.14**cyclic**

term used to describe events which repeat in a regular and repetitive manner

3.4.15**deafness**

situation where the node can not hear the moderator DLPDU but can hear other link traffic

**3.4.16
device**

physical hardware connection to the link

NOTE A device may contain more than one node.

**3.4.17
DLPDU**

data-link protocol data unit

NOTE A DLPDU consists of a source MAC ID, zero or more Lpackets and an FCS as transmitted or received by an associated PhE.

**3.4.18
end delimiter**

unique set of M_symbols that identifies the end of a DLPDU

**3.4.19
error**

discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

**3.4.20
extended link**

links connected by DL-routers (sometimes called a local area network)

**3.4.21
FCS error**

error that occurs when the computed frame check sequence value after reception of all the octets in a DLPDU does not match the expected residual

**3.4.22
fixed tag**

two octet identifier (tag) which identifies a specific service to be performed by either

- a) that receiving node on the local link which has a specified MAC ID, or
- b) all receiving nodes on the local link

NOTE Identification of the target node(s) is included in the two octet tag.

**3.4.23
frame**

denigrated synonym for DLPDU

**3.4.24
generic tag**

three octet identifier (tag) which identifies a specific piece of application information

**3.4.25
guardband**

time slot allocated for the transmission of the moderator DLPDU

**3.4.26
implicit token**

mechanism that governs the right to transmit

NOTE No actual token message is transmitted on the medium. Each node keeps track of the MAC ID of the node that it believes currently holds the right to transmit. The right to transmit is passed from node to node by keeping a record of the node that last transmitted. A slot time is used to allow a missing node to be skipped in the rotation.

3.4.27**implicit token register**

register that contains the MAC ID of the node that holds the right to transmit

3.4.28**instance**

actual physical occurrence of an object within a class, identifying one of many objects within the same object class

EXAMPLE California is an instance of the object class state.

NOTE The terms object, instance, and object instance are used to refer to a specific instance.

3.4.29**instance attribute**

attribute that is unique to an object instance and not shared by the object class

3.4.30**instantiated**

object that has been created in a device

3.4.31**Keeper**

object responsible for distributing link configuration data to all nodes on the link

3.4.32**link**

collection of nodes with unique MACTDs, attached to Ph-segments connected by Ph-repeaters

3.4.33**little endian**

model of memory organization which stores the least significant octet at the lowest address, and of transmission organization which transfers the least significant octet first on the medium

3.4.34**Lpacket**

well-defined sub-portion of a DLPDU consisting of

- a) size and control information
- b) a fixed tag or a generic tag, and
- c) DLS-user data or, when the tag has DL-significance, DL-data

3.4.35**M_symbol(s)**

symbols that represent the data bits and related information encoded and transmitted by the PhL

3.4.36**maximum scheduled node**

node with highest MAC ID that can use scheduled time on a link

3.4.37**maximum unscheduled node**

node with highest MAC ID that can use unscheduled time on a link

3.4.38**Message Router**

object within a node that distributes messaging requests to the appropriate application objects

3.4.39

moderator

node with the lowest MAC ID that is responsible for transmitting the moderator DLPDU

3.4.40

moderator DLPDU

DLPDU transmitted by the node with the lowest MAC ID for the purpose of synchronizing the nodes and distributing the link configuration parameters

3.4.41

multipoint connection

connection from one node to many

NOTE Multipoint connections allow messages from a single producer (publisher) to be received by many consumer (subscriber) nodes.

3.4.42

network

series of nodes connected by some type of communication medium. The connection paths between any pair of nodes can include repeaters, routers and gateways

3.4.43

network access monitor (NAM)

component within a node that manages communication with a temporary node connected to the nodes local NAP

3.4.44

network access port (NAP)

PhL variant that allows a temporary node to be connected to the link by connection to the NAP of a permanent node

3.4.45

network address or node address

node's address on the link (also called MAC ID)

3.4.46

network status indicators

indicators on a node indicating the status of the Physical and Data Link Layers

3.4.47

network update time (NUT)

repetitive time interval in which data can be sent on the link

3.4.48

node

logical connection to a local link, requiring a single MAC ID

NOTE A single physical device may appear as many nodes on the same local link. For the purposes of this protocol, each node is considered to be a separate DLE.

3.4.49

non-concurrence

situation where a transmission is received from an unexpected MAC ID, which appears to violate the time based access protocol

NOTE This may occur when a connection is made between two working links that are not synchronized with each other but who have the same configuration information.

3.4.50

non-data symbol

PhL symbol which violates the requirements of Manchester biphase L encoding

3.4.51**object**

abstract representation of a particular component within a device

NOTE An object can be

- 1) an abstract representation of a computer's capabilities. Objects can be composed of any or all of the following components:
 - a) data (information which changes with time);
 - b) configuration (parameters for behavior);
 - c) methods (things that can be done using data and configuration).
- 2) a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behavior.

3.4.52**object specific service**

service defined by a particular object class to perform a required function which is not performed by a common service

NOTE An object specific service is unique to the object class which defines it.

3.4.53**originator**

client responsible for establishing a connection path to the target

3.4.54**permanent node**

node whose connection to the network does not utilize the network access port (NAP) PhL variant

NOTE This node may optionally support a NAP PhL variant to allow temporary nodes to connect to the network.

3.4.55**produce**

act of sending data to be received by a consumer

3.4.56**producer**

node that is responsible for sending data

3.4.57**redundant media**

system using more than one medium to help prevent communication failures

3.4.58**requested packet interval (RPI)**

measure of how frequently the originating application requires the transmission of Lpackets or data packets from the target application

3.4.59**rogue**

node that has received a moderator DLPDU that disagrees with the link configuration currently used by this node

3.4.60**scheduled**

data transfers that occur in a deterministic and repeatable manner on predefined NUTs

3.4.61**server**

object which provides services to another (client) object

3.4.62

service

operation or function than an object and/or object class performs upon request from another object and/or object class

NOTE A set of common services is defined and provisions for the definition of object-specific services are provided. Object-specific services are those which are defined by a particular object class to perform a required function which is not performed by a common service.

3.4.63

slot time

maximum time required for detecting an expected transmission

NOTE Each node waits a slot time for each missing node during the implied token pass.

3.4.64

start delimiter

unique set of M_symbols that identifies the beginning of a DLPDU

3.4.65

supernode

DLE with a MAC ID of zero

NOTE This DLE DL-address is reserved for special DLL functions.

3.4.66

table unique identifier (TUI)

unique reference code used by ControlNet and Keeper objects to reference a consistent set of link configuration attribute

3.4.67

tag

shorthand name for a specific piece of application information, two or three octets in length

3.4.68

target

end-node to which a connection is established

3.4.69

temporary node

same as transient node

3.4.70

tMinus

number of NUTs before a new set of link configuration parameters are to be used

3.4.71

tone

instant of time which marks the boundary between two NUTs

3.4.72

tool

executable software program which interacts with the user to perform some function

EXAMPLE Link scheduling software (LSS).

3.4.73

transaction id

field within a UCMM header that matches a response with the associated request, which a server echoes in the response message

3.4.74**transient node**

node that is only intended to be connected to the network on a temporary basis using the NAP PhL medium connected to the NAP of a permanent node

3.4.75**Unconnected Message Manager (UCMM)**

component within a node that transmits and receives unconnected explicit messages and sends them directly to the Message Router object

3.4.76**unconnected service**

messaging service which does not rely on the set up of a connection between devices before allowing information exchanges

3.4.77**unscheduled**

data transfers that use the remaining allocated time in the NUT after the scheduled transfers have been completed

3.5 Type 2 symbols and abbreviations

ACM	Access control machine	
CA	Clock accuracy	
CAN	Controller area network	[ISO 11898:1993]
COP	Connection originator password	
IP	Internet Protocol	
LED	Light emitting diode.	
LSS	Link scheduling software	
MAC ID	MAC address of a node	
NAM	Network attachment monitor	
NAP	Network access port	
ND	Non-data symbol	
NUT	Network update time	
PT	Programming terminal (a temporary network connection)	
Rcv	Receive	
RPI	Requested Lpacket interval	
Rx	Receive	
RxLLC	Receive logical link control	
RxM	Receive machine	
SEM	State event matrix	
SMAX	MAC ID of the maximum scheduled node	
STD	State transition diagram, used to describe object behavior	
TCP	Terminal Control Protocol	
TUI	Table unique identifier	
Tx	Transmit	
TxLLC	Transmit logical link control	
TxM	Transmit machine	
UCCM	Unconnected Message Manager	
UMAX	MAC ID of maximum unscheduled node	
USR	Unscheduled start register	

4 Overview of the DL-protocol

4.1 General

4.1.1 DLL architecture

The Type 2 DLL is modeled as an integrated Access Control Machine (ACM), with scheduling support functions, designed for reliable and efficient support of higher-level connection-mode and connectionless data transfer services. Interoperating with these higher level functions are DLL management functions.

PhL framing and delimiters are managed by DLL functions for serializing and deserializing M_symbol requests and indications.

Within the Data Link Layer, the Access Control Machine (ACM) has the primary responsibility for detecting and recovering from network disruptions. The major objectives of the ACM are

- to make sure that the local node detects and fully utilizes its assigned access slots in the schedule;
- to make sure that the local node does not interfere with the transmissions of other nodes, especially the moderator node;
- to start the next NUT (see 4.1.2) on schedule, whether the moderator DLPDU is heard or not;
- if the local node is the moderator, to transmit each moderator DLPDU strictly on schedule.

The Data Link Layer is comprised of the components listed in Table 1 :

Table 1 – Data-link layer components

Components	Description
Access Control Machine (ACM)	receives and transmits control DLPDUs and header information, and determines the timing and duration of transmissions.
Transmit LLC (TxLLC)	buffers DLSdUs received from Station Management and the DLS-user, and determines which should be transmitted next.
Receive LLC (RxLLC)	performs the task of quarantining received link units until they are validated by a good FCS.
Transmit Machine (TxM)	receives requests to send DLPDU headers and trailers, and Lpackets from the ACM, and breaks them down into octet symbol requests that are sent to the Serializer.
Receive Machine (RxM)	assembles received Lpackets from octet symbols received from the Deserializer, and submits them to the RxLLC.
Serializer	receives octet symbols, encodes and serializes them, and sends them as M_symbols to the PhL. It is also responsible for generating the FCS.
Deserializer	receives M_symbols from the PhL, converts M_symbols into octets and sends them to the receive machine. It is also responsible for checking the FCS.
DLL Management Interface	holds the Station Management variables that belong to the DLL, and helps manage synchronized changes of the link parameters.

The internal arrangement of these components, and their interfaces, are shown in Figure 2. The arrowheads illustrate the primary direction of flow of data and control.

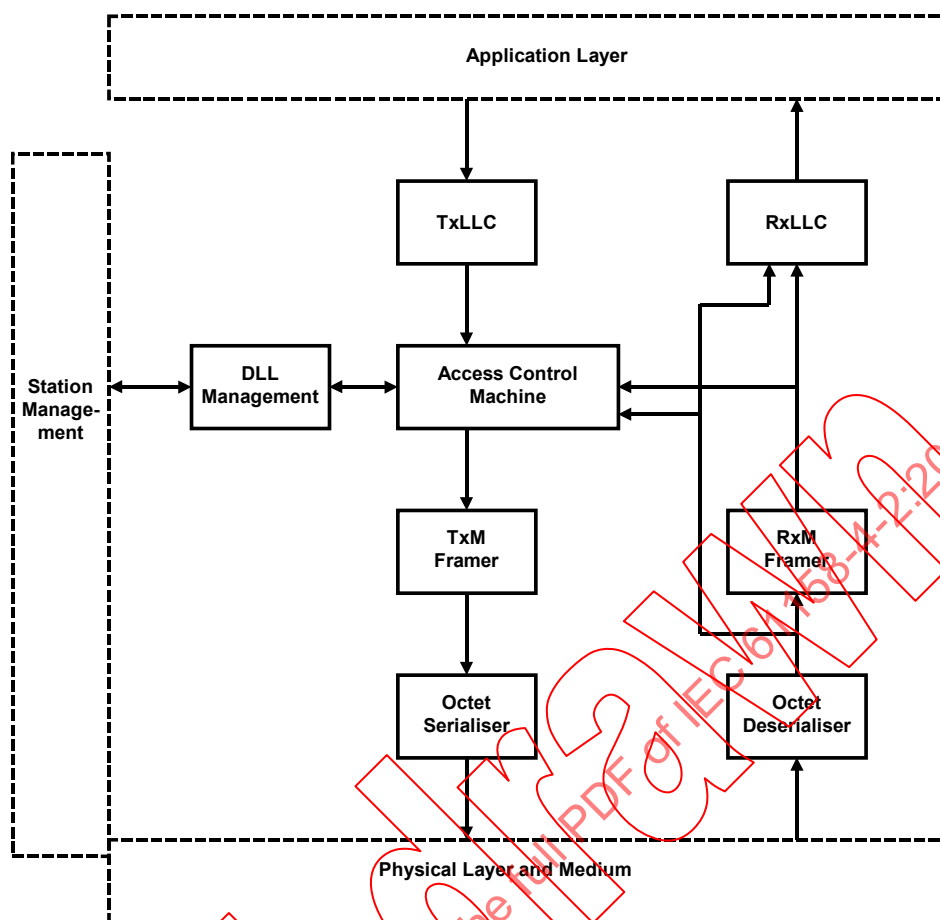


Figure 2 – Data-link layer internal architecture

4.1.2 Access control machine (ACM) and scheduling support functions

The ACM functions schedule all communication from one DLE to another. The timing of this communication are regulated at two levels,

- 1) to provide an accurate time based distribution of scheduled opportunities for designated communications in accord with, and with a pre-established schedule for, the local link and any extended network of links.
- 2) to provide a democratic distribution of unscheduled opportunities for arbitrary communications, generally in a cyclic but asynchronous manner.

Accurate schedule timing as provided by 1) is very important to support many control and data collection tasks in the applications domain of this protocol. The schedule is based on a fixed, repetitive time cycle called network update time (NUT). The NUT is maintained in close synchronism among all nodes participating in the DLL and used to provide two types of access opportunity;

- 1) One opportunity for each active station to transfer one scheduled DLPDU containing multiple scheduled DLSDUs.

NOTE 1 Only DLSDUs with scheduled as their QoS parameter may be included in a scheduled DLPDU.

NOTE 2 DLSDUs with scheduled as their QoS parameter are usually connection mode transfers with generic as their tag parameter.

- 2) One opportunity for at least one active station to send one background (unscheduled) DLPDU containing multiple DLSDUs of any QoS type. If time is available within a NUT, one background MAC opportunity is offered to each other active station in order of their MAC ID address. The station MAC ID for the first background opportunity in each NUT, is incremented (+1 modulo the set of background addresses) for each successive NUT to democratically share the available time among active stations.

Support procedures are included for automatic transfer to backup scheduling services and to manage the use of redundant Ph media.

4.1.3 Connection-mode, connectionless-mode data transfer and DL service

The connection mode, connectionless mode and DL services provide the necessary functionality for

- managing DL provider interactions with the DL user by converting request and response primitives into necessary DLE operations and generating indications and confirm primitives where appropriate,
- determination of the DL address and control information to be added to each DLSDU,
- local confirmation of status for each outgoing DLSDU,
- DLPDU formation by concatenation of multiple DL user DLSDUs for efficient transfer as single PhPDUs, subject to DLPDU size limitations, QoS parameters and the type of access opportunity.

4.2 Services provided by the DL

4.2.1 Overview

The DLL provides connectionless data transfer services and connection-mode data transfer services for limited-size DLSDUs, an internally synchronized time service, scheduling services to control the time allocation of the underlying shared PhL service, a DL(SAP)-address and queue management service, and a packing/unpacking service that combines multiple DLSDUs into a single data transfer DLPDU for efficient use of each access opportunity.

4.2.2 QoS

4.2.2.1 Definition

The QoS parameter specifies priority options for DLSDUs and access opportunities. The available values are as follows:

4.2.2.2 DLL scheduled priority

The scheduled priority QoS provides accurate time based cyclic and acyclic sending of DLSDUs. The execution timing for this fieldbus-scheduled service can be accurate and repeatable to better than 1 ms. Scheduled priority is normally used with connected mode services.

NOTE The master keeper is responsible for regular publication of a unique table identifier (TUI) which ensures all nodes have a reference for the current set of link operating parameters. This TUI publication uses a fixed tag and is sent with QoS priority set to scheduled.

4.2.2.3 DLL high priority (unscheduled)

The high priority QoS provides acyclic sending of DLSDUs with a bounded upper time for the sending delay. Data on this priority is sent only when all scheduled data has been sent and a non-scheduled sending opportunity is available.

4.2.2.4 DLL low priority (unscheduled)

The low priority QoS provides for sending of DLSDUs only on an as-available basis. Data on this priority is sent only when all other data priorities have been sent and a non-scheduled sending opportunity is available.

NOTE High and Low priority are used only in a local sense to set the order of servicing among locally submitted, unscheduled DLS-user-data.

4.3 Structure and definition of DL-addresses

4.3.1 General

DL-addresses are used as MAC ID addresses (Link node designators), Fixed tag addresses (DLSAP-addresses and group DL-addresses), and Generic tag addresses (DLCEP-addresses).

This subclause defines the form and structure of DL-addresses and includes specific definitions for some standard DL-addresses.

All addresses are formed of octets. Each octet shall be transferred to the Ph layer low-bit first and multiple octets shall be transferred low-order octet first (little endian format).

Three types of DL addresses are used.

4.3.2 MAC ID address

MAC ID addresses identify physical nodes on the local link as shown in Figure 3.

Permitted values are within the range 0–99, values from 100–254 are reserved.

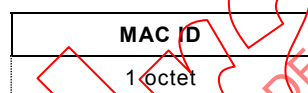


Figure 3 – Basic structure of a MAC ID address

Table 2 specifies the use of MAC ID addresses.

Table 2 – MAC ID addresses allocation

MAC ID	Meaning and purpose
0x0	temporary address used for maintenance
SMAX	current maximum address value for Scheduled access opportunities. SMAX ≤ UMAX
UMAX	current maximum address value for Unscheduled access opportunities SMAX. ≤ UMAX ≤ 0x63
0x64 to 0xFE	reserved
0xFF	broadcast address to all MAC ID nodes on this link

4.3.3 Generic tag address

Generic tag addresses types identify all data transfers for connected mode services. The available capacity in the DLL is 3 octets, see Figure 4.

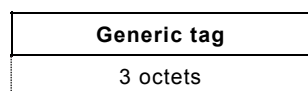


Figure 4 – Basic structure of a generic tag address

NOTE 1 Generic tags are used to send connected DLSDUs and may be associated with any of the available QoS priorities (Scheduled, High, Low).

NOTE 2 Negotiation and management of generic tags and connected mode services are the responsibility of the DL user.

4.3.4 Fixed tag address

Fixed tag addresses types identify DLSAPs within a designated station MAC ID. They are used with all data transfers for connectionless mode services.

NOTE Fixed tags are used to send connectionless DLSDUs and are usually associated with unscheduled QoS priorities (High, Low). Typical uses of fixed tags are for station management and to establish connections.

Each fixed tag address shall consist of two octets as shown in Figure 5.

Fixed tag service	Destination MAC ID
1 octet	1 octet

Figure 5 – Basic structure of a fixed tag address

The first octet shall specify the service access point in the destination node as specified in Table 3. The second octet shall be the address of the destination node. The destination address shall be either a MAC ID or 0xFF, which indicates the broadcast address to all MAC IDs on the local link.

Table 3 – Fixed tag service definitions

Fixed tag service	Meaning
0x00	moderator
0x01 - 0x08	vendor specific
0x09	ping request
0x0A - 0x14	vendor specific
0x15	tlMinus
0x16 - 0x28	vendor specific
0x29	ping reply
0x2A - 0x3F	vendor specific
0x40 - 0x6F	reserved
0x70 - 0x7F	vendor-specific
0x80	I'm alive
0x81	link parameters
0x82	reserved
0x83	UCMM
0x84	TUI
0x85	IP (reserved for Internet Protocol)
0x86	WAMI
0x87	reserved
0x88	Keeper UCMM
0x89	Ethernet (used for address resolution protocol)
0x8A - 0x8B	reserved
0x8C	Time distribution
0x8D - 0x8F	reserved for time distribution
0x90	debug
0x91 - 0xCF	reserved
0xD0 - 0xEF	group addresses
0xF0 - 0xFF	vendor specific

Fixed tags in the range 0xD0 to 0xEF shall be reserved to permit group screening of connection identifiers.

4.4 Services assumed from the PhL

4.4.1 General requirements

The following subclauses include requirements for interface to a Type 2 PhL that are not required for other Types.

- Transmit Machine.
- Receive Machine.
- Serializer.
- Deserializer.

This is necessary because the Ph-Interface Data Units (PhIDUs) that Type 2 passes across the DLL–PhL interface are individual serial data and non-data symbols, designated on the DLE side as M_Symbols (see Table 4), whereas PhIDUs for other Types are not conceptualized in this way. As a consequence, a Type 2 Data Link Layer requires a Type 2 PhL.

4.4.2 Data encoding rules

The M_Symbols transferred at the DLL to PhL interface are of four types designated 0, 1, +, -. They will be encoded inside the PhL into appropriate Ph_Symbols as shown in Table 4. The M_0 and M_1 will be encoded into Ph_Symbols that represent Manchester biphas L data encoding rules. The M_ND symbols are non-data symbols to allow for the creation of unique data patterns used for start and end delimiters. The example of signal voltage waveform as it might be applied to an electrically-conductive medium is shown in an idealized form in Figure 6 to provide an example of the data encoding rules shown in Table 4.

Table 4 – Data encoding rules

Data bits (common name)	M_Symbol representation	Ph_Symbol encoding	Manchester encoded
data “zero”	M_0 or {0}	{L,H}	0
data “one”	M_1 or {1}	{H,L}	1
“non_data+”	M_ND+ or {+}	{H,H}	no data
“non_data-”	M_ND- or {-}	{L,L}	no data

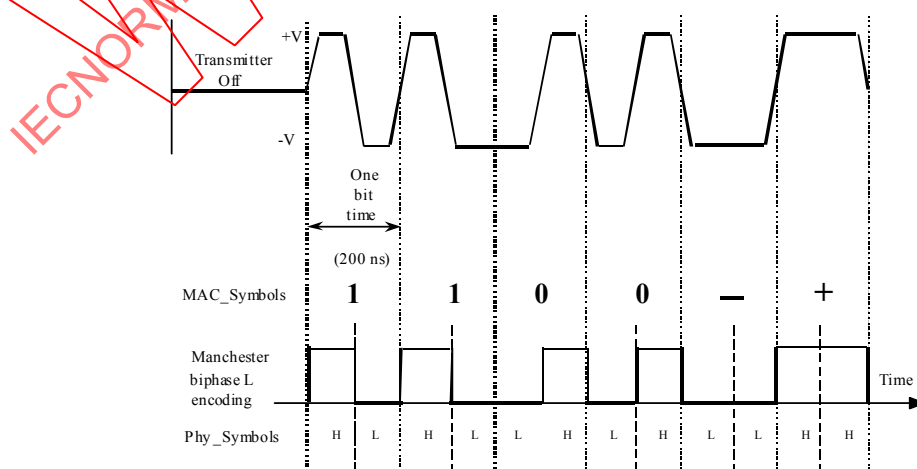


Figure 6 – M_symbols and Manchester encoding at 5 MHz (informative)

4.4.3 DLL to PhL interface

4.4.3.1 General

The DLL to PhL interface need not be exposed in the implementation of any PhL variant. This interface may be internal to the node. If conformance to the DLL to PhL interface is claimed, it shall conform to the requirements of this clause.

4.4.3.2 Ph_lock_indication

ph_lock_indication shall provide an indication of either data lock or Ph_Symbol synchronization by the PhL. Valid states for ph_lock_indication shall be true and false. ph_lock_indication shall be true whenever valid Ph_Symbols are present at the PhL interface and the PhL to DLL interface timing of M_Symbols conforms to the PhL requirements. It shall be false between frames (when no Ph_Symbols are present on the medium) or whenever data synchronization is lost or the timing fails to conform to the PhL requirements. ph_lock_indication shall be true prior to the beginning of the start delimiter.

4.4.3.3 Ph_frame_indication

ph_frame_indication shall provide an indication of a valid data frame from the PhL. Valid states for ph_frame_indication shall be true and false. ph_frame_indication shall be true upon ph_lock_indication = true and reception of the first valid start delimiter. ph_frame_indication shall be false at reception of next M_ND symbol (following the start delimiter) or ph_lock_indication = false.

NOTE This signal provides octet synchronization to the DLL.

4.4.3.4 Ph_carrier_indication

ph_carrier_indication shall represent the presence of a signal carrier on the medium. The ph_carrier_indication shall be true if internal PhL identification of signal carrier has been true during any of the last 4 M_symbol times and it shall be false otherwise.

4.4.3.5 Ph_data_indication

ph_data_indication shall represent the M_Symbols that are decoded from the PhL internal representations such as those shown in Table 4. Valid M_symbols shall be M_0 {0}, M_1 {1}, M_ND+ {+}, and M_ND- {-} as shown in Table 5.

Table 5 – M Data symbols

Data bits (common name)	M_Symbol representation
data "zero"	M_0 or {0}
data "one"	M_1 or {1}
"non_data+"	M_ND+ or {+}
"non_data–"	M_ND– or {–}

The ph_data_indication shall represent the M_Symbols as decoded within the PhL whenever ph_lock_indication is true.

4.4.3.6 Ph_status_indication

ph_status_indication shall represent the delimiter status of the frame which was received from the PhL as shown in Table 6. Valid symbols shall be Normal, Abort, and Invalid. ph_status_indication shall indicate Normal after reception of a frame (ph_frame_indication = true) composed of a start delimiter, valid data (no M_ND symbols) and an end delimiter. ph_status_indication shall indicate Abort after reception of a frame (ph_frame_indication = true) composed of a start delimiter, valid data, and a second start delimiter. ph_status_indication

shall indicate Invalid after reception of a frame (`ph_frame_indication = true`) composed of a start delimiter and the detection of any M_ND symbol which was not part of a start or end delimiter.

Table 6 – Truth table for `ph_status_indication`

<code>Ph_status_indication</code>	<code>Ph_frame_indication</code>	Start delimiters in a single frame	End delimiter detection	Any non-delimiter Manchester violations
Normal	true	1	true	false
Abort	true	2	don't care	false
Invalid	true	1	don't care	true

4.4.3.7 `Ph_data_request`

`ph_data_request` shall present the M_Symbols to be transmitted. Valid symbols shall be M_0, M_1, M_ND+ or M_ND– as shown in Table 5. `ph_data_request` shall indicate M_0 when no data is to be transmitted (and `ph_frame_request = false`).

4.4.3.8 `Ph_frame_request`

`ph_frame_request` shall be true when `ph_data_request` presents M_Symbols to be encoded into the appropriate Ph_Symbols by the PhL, and shall be false when no valid M_Symbols are to be transferred to the PhL.

4.5 Functional classes

Implementations of this protocol fall into two broad classes:

- connection originators (sometimes called Scanners);
- connection responders or connection targets (sometimes called Adaptors).

Devices with connection origination capability will include functions such as Scheduling object, support for external Configuration Tools and usually a Keeper object.

Connection responders are typically simpler devices which do not need the full set of capabilities.

This specification does not mandate particular groupings of functions for classes of implementation devices.

NOTE Grouping of services and functions into implementation classes may be done by separate specifications for profiles.

5 General structure and encoding of PhIDUs and DLPDUs and related elements of procedure

5.1 Overview

The DLL and its procedures are those necessary to provide the services offered to DLS user by using the services available from the PhL. The main services of this protocol are

- a) provision of timely access opportunities to support scheduled data-transfer DL-services on one link and on an extended multi-link network;
- b) the packing of multiple DLS-user transfers into a single PhL transfer unit (DLPDU) to efficiently use each transfer opportunity;
- c) the efficient distribution of DLS-user data from a publishing DLS-user to a set of subscribing DLS-users;
- d) the provision of a synchronized sense of internal time among the DLEs and available to the DLS-users of the extended link;
- e) the standardized availability of local DL-address, buffer and queue management capabilities.

5.2 Media access procedure

The primary task of the Data Link Layer is to determine, in co-operation with other Data Link Layers on the same link, the granting of permission to transmit on the medium. At its upper interface, the DLL provides services to receive and deliver service data units for the DLS user and Station Management.

The DLL protocol is based on a fixed, repetitive time cycle called a network update time (NUT). The NUT is maintained in close synchronism among all nodes on the link. A node is not permitted access to transmit on the medium if its NUT does not agree with the NUT currently being used on a link wide basis. Different links may have different NUT duration.

Each node contains its own timer synchronized to the NUT for the local link. Media access is determined by local sub-division of the NUT into access slots. Access to the medium is in sequential order based on the MAC ID of the node. Specific behaviors have been incorporated into the access protocol allowing a node which temporarily assumes a MAC ID of zero to perform link maintenance. The MAC ID numbers of all nodes on a link are unique. Any DLL detecting the presence of a duplicate MAC ID shall immediately stop transmitting.

An implicit token passing mechanism is used to grant access to the medium. Each node monitors the source MAC ID of each DLPDU received. At the end of a DLPDU, each DLL sets an "implicit token register" to the received source MAC ID + 1. If the implicit token register is equal to the local MAC ID, then the node shall transmit one DLPDU containing one or more Lpackets with data or a null DLPDU. In all other cases, the node watches for either a new DLPDU from the node identified by the "implicit token register" or a time-out value if the identified node fails to transmit. In each case, the "implicit token" is automatically advanced to the next MAC ID. All nodes have the same value in their "implicit token register" preventing collisions on the medium.

The time-out period (called the "slot time") is based on the amount of time required for

- the current node to hear the end of the transmission from the previous node;
- the current node to begin transmitting;
- the next node to hear the beginning of the transmission from the current node.

The slot time is adjusted to compensate for the total length of the medium since the propagation delay of the medium effects the first and last item on the previous list.

NOTE The calculation of slot time is specified in 8.2.

Each NUT is divided into three major parts: scheduled, unscheduled, and guardband as shown in Figure 7. This sequence is repeated in every NUT. The implicit token passing mechanism is used to grant access to the medium during both the unscheduled and scheduled intervals.

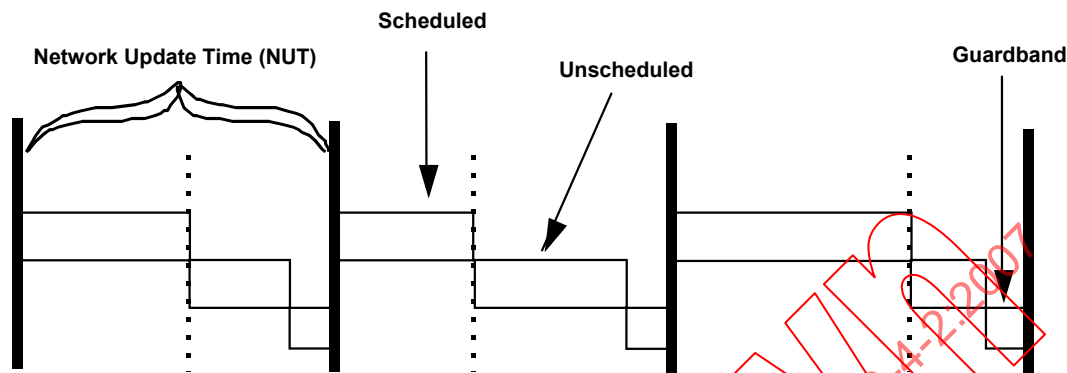


Figure 7 – NUT structure

During the scheduled part of the NUT, each node, starting with node 0 and ending with node SMAX, gets a chance to transmit time-critical (scheduled) data. SMAX is the MAC ID of the highest numbered node that has access to the medium during the scheduled part of the NUT. Every node between 0 and SMAX has only one opportunity to send one DLPDU of scheduled data in each NUT. The opportunity to access the medium during the scheduled time is the same for each node in every NUT. This allows data that is transmitted during the scheduled portion of the NUT to be sent in a predictable and deterministic manner. Figure 8 shows how the permission to transmit is granted during the scheduled time. The DLS-user regulates the amount of data that each node may transmit during this scheduled token pass.

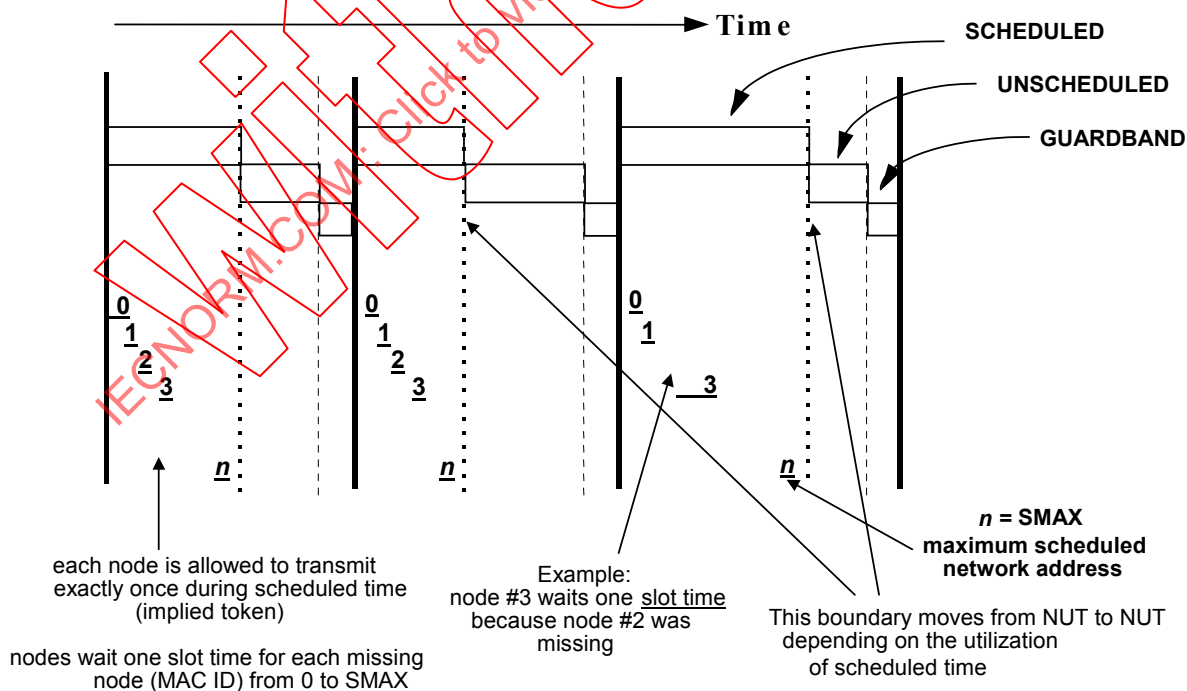


Figure 8 – Media access during scheduled time

During the unscheduled part of the NUT, each node from 0 to UMAX shares the opportunity to transmit one DLPDU of non-time critical data in a round robin fashion, until the allocated NUT duration is exhausted. UMAX is the MAC ID of the highest numbered node that has access to

the medium during the unscheduled part of the NUT. The round robin method of access opportunity enables every node between 0 and UMAX to have zero, one or many opportunities to send unscheduled data depending on how much of the NUT remains after the completion of the scheduled time. Variations in scheduled traffic means the opportunity to access the medium during the unscheduled time may be different for each node in every NUT. Figure 9 shows how the permission to transmit is granted during the unscheduled time. The MAC ID of the node that goes first in the unscheduled part of the NUT is incremented by 1 for each NUT. The unscheduled token begins at the MAC ID specified in the unscheduled start register (USR) of the previous moderator DLPDU. The USR increments by one modulo (UMAX+1) each NUT. If the USR reaches UMAX before the guardband, it returns to zero and the token pass continues.

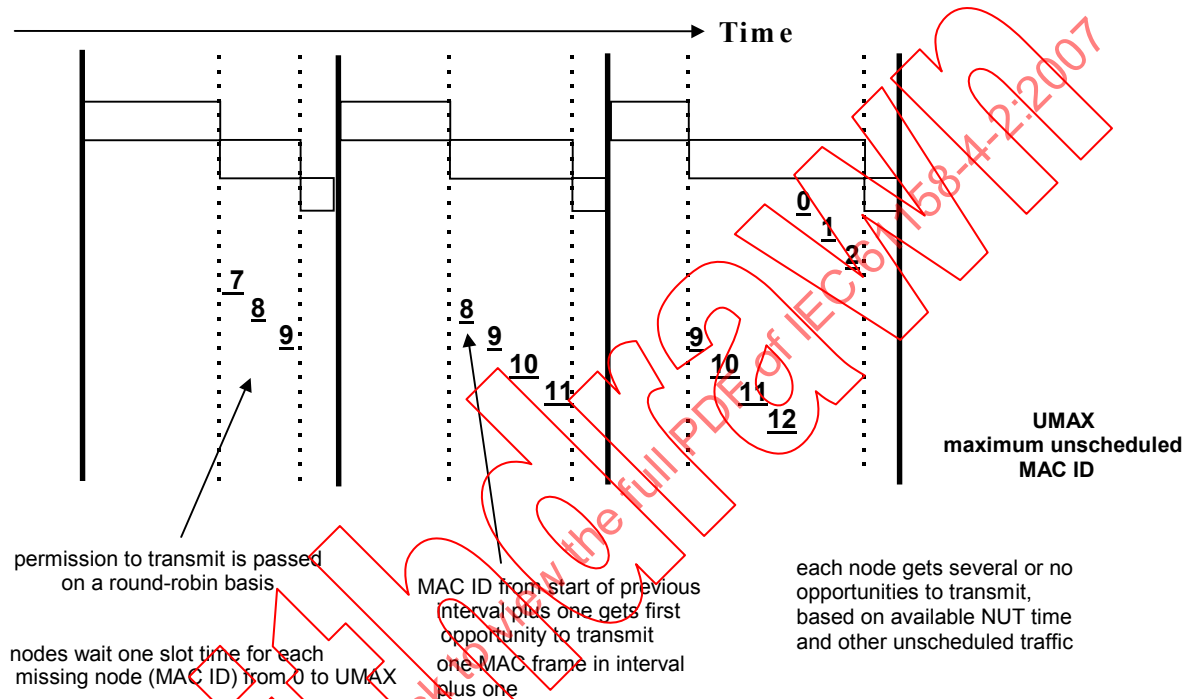


Figure 9 – Media access during unscheduled time

When the guardband is reached, all nodes stop transmitting. A node is not allowed to start a transmission unless it can be completed before the beginning of the guardband. During the guardband, the node with the lowest MAC ID (called the “moderator”) transmits a maintenance message (called the “moderator DLPDU”) that accomplishes two things:

- it keeps the NUT timers of all nodes synchronized;
- it publishes critical link parameters enabling all members of the local DLL group to share a common version of important DLL values such as NUT, slot time, SMAX, UMAX, etc.

The moderator transmits the moderator DLPDU, which re-synchronizes all nodes and restarts the NUT. Following the receipt of a valid moderator DLPDU, each node compares its internal values with those transmitted in the moderator DLPDU. A node using link parameters that disagree with the moderator disables itself. If the moderator DLPDU is not heard for 2 consecutive NUTs, the node with the lowest MAC ID assumes the moderator role, and begins transmitting the moderator DLPDU in the guardband of the third NUT. A moderator node that notices another node on-line and transmitting with a MAC ID lower than its own immediately cancels its moderator role.

Typical situations that may cause disruption of the DLL access protocol include

- induced noise on the link;
- poor cable or termination quality;

— physically connecting two links together while the network is operating.

One common consequence of such disruption is that nodes may be caused to disagree as to which node should be transmitting; this is called a network “non-concurrence”. Another potential problem occurs when the nodes do not agree to the same link configuration parameters. A node that disagrees with the link parameters as transmitted by the moderator is called a “rogue” and immediately stops transmitting. The DLL access protocol is designed to recover a rogue node and bring it back on-line.

5.3 DLPDU structure and encoding

5.3.1 General

DLPDUs are the DLPDUs passed to the Ph layer for sending on the Ph medium. The DLPDUs are formed by the transmit logical link control (TxLLC) protocol in two stages:

- DLSDUs are formed into Link-units (Lpackets);
- Multiple Lpackets are packed into one DLPDU subject to constraints set by the QoS required, the type of access opportunity available and limits on DLPDU overall length.

5.3.2 DLPDU components

The general DLPDU structure is shown in Figure 10 and managed by the DLL support units TxFramer, RxFramer, Octet Serializer and Octet Deserializer as shown in Figure 2.

The components of the DLPDU shall be transmitted in the following order: preamble, start delimiter, source MAC ID, zero or more Lpackets, FCS and end delimiter.

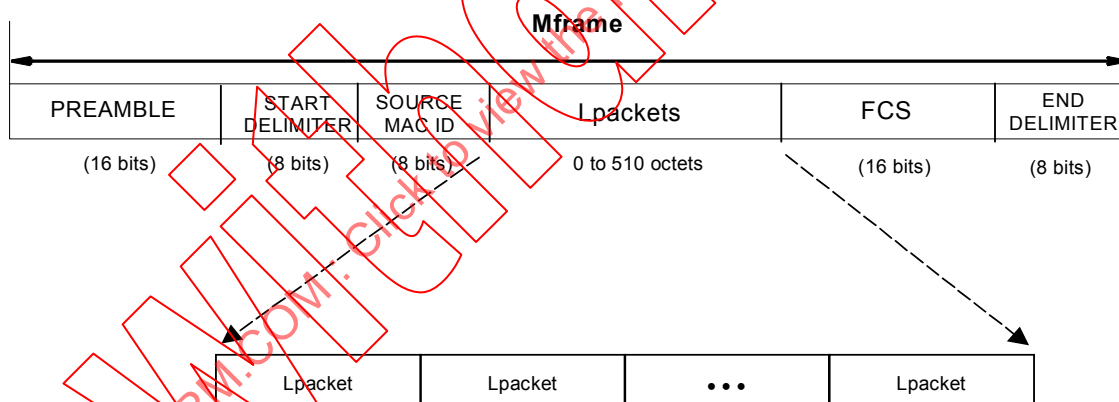


Figure 10 – DLPDU format

5.3.3 Preamble

The preamble shall be composed of 16 consecutive {1} M_Symbols.

5.3.4 Start and end delimiters

The start delimiter shall consist of these M_Symbols, {+, 0, –, +, –, 1, 0, 1}, transmitted from left to right.

The end delimiter shall consist of these M_Symbols, {1, 0, 0, 1, +, –, +, –}, transmitted from left to right.

The use of non-data M_Symbols shall be reserved for the start and end delimiters.

5.3.5 DLPDU octets and ordering

The source MAC ID, Lpackets, and FCS shall be composed of octets. An octet shall be composed of exactly eight M_Symbols each of which shall be either {0} or {1}. An octet shall be transferred to the PhL low bit first. Multiple octet fields shall be transmitted low order octet first (little endian format).

5.3.6 Source MAC ID

The source MAC ID address shall be a single octet as specified in 4.3.2. A node may also temporarily assume a MAC ID of zero to perform link maintenance.

5.3.7 Lpackets field

Zero or more Lpackets shall be concatenated into a single DLPDU subject to the constraint that the maximum number of octets which compose all Lpackets in a DLPDU shall be 510.

An additional constraint is that DLPDUs sent during a scheduled access opportunity shall only contain Lpackets with a QoS parameter of scheduled.

5.3.8 Frame check sequence (FCS)

5.3.8.1 Frame check sequence requirements

The FCS shall be a modified CCITT checksum using the 16 bit polynomial: $G(X) = X^{16} + X^{12} + X^5 + 1$. A two octet FCS shall be included in each DLPDU.

The formal concepts for its generation and checking on reception are described in 5.3.8.2. The generation and checking of the FCS are further specified in 9.7 and 9.8.

5.3.8.2 Frame check sequence formal concepts

5.3.8.2.1 Frame check sequence field

Within this subclause, any reference to bit K of an octet is a reference to the bit whose weight in a one-octet unsigned integer is 2^K .

NOTE This is sometimes referred to as "little endian" bit numbering.

For this standard, as in other International Standards (for example, ISO/IEC 3309, ISO/IEC 8802 and ISO/IEC 9314-2), DLPDU-level error detection is provided by calculating and appending a multi-bit frame check sequence (FCS) to the other DLPDU fields during transmission to form a "systematic code word"³ of length n consisting of k DLPDU message bits followed by $n - k$ (equal to 16) redundant bits, and by calculating during reception that the message and concatenated FCS form a legal (n, k) code word. The mechanism for this checking is as follows, where the generic form of the generator polynomial for this FCS construction is specified in equation (4) and the polynomial for the receiver's expected residue is specified in equation (9). The specific polynomials for this standard are specified in Table 7.

³ W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes* (2nd edition), MIT Press, Cambridge, 1972.

Table 7 – FCS length, polynomials and constants

Item	Value
$n-k$	16
$G(X)$	$X^{16} + X^{12} + X^5 + 1$ (notes 1, 2)
$R(X)$	$X^{12} + X^{11} + X^{10} + X^8 + X^3 + X^2 + X + 1$ (notes 2, 3)
NOTE 1 Code words $D(X)$ constructed from this $G(X)$ polynomial have Hamming distance 4 for lengths $\leq 4\,095$ octets, and all errors of odd weight are detected.	
NOTE 2 These are the same polynomials and method as specified in ISO/IEC 3309 (HDLC).	
NOTE 3 The remainder $R(x)$ should be 0001 1101 0000 1111 (X^{15} to X^0 , respectively) in the absence of errors.	

5.3.8.2.2 At the sending DLE

The original message (that is, the DLPDU without an FCS), the FCS, and the composite message code word (the concatenated DLPDU and FCS) shall be regarded as vectors $M(X)$, $F(X)$, and $D(X)$, of dimension k , $n - k$, and n , respectively, in an extension field over $GF(2)$. If the message bits are $m_1 \dots m_k$ and the FCS bits are $f_{n-k-1} \dots f_0$, where

$m_1 \dots m_8$ form the first octet sent,

$m_{8N-7} \dots m_{8N}$ form the N th octet sent,

$f_7 \dots f_0$ form the last octet sent, and

m_1 is sent by the first PhL symbol(s) of the message and f_0 is sent by the last PhL symbol(s) of the message (not counting PhL framing information),

NOTE 1 This “as transmitted” ordering is critical to the error detection properties of the FCS.

then the message vector $M(X)$ shall be regarded to be

$$M(X) = m_1 X^{k-1} + m_2 X^{k-2} + \dots + m_{k-1} X^1 + m_k \quad (1)$$

and the FCS vector $F(X)$ shall be regarded to be

$$\begin{aligned} F(X) &= f_{n-k-1} X^{n-k-1} + \dots + f_0 \\ &= f_{15} X^{15} + \dots + f_0 \end{aligned} \quad (2)$$

The composite vector $D(X)$, for the complete DLPDU, shall be constructed as the concatenation of the message and FCS vectors

$$\begin{aligned} D(X) &= M(X) X^{n-k} + F(X) \\ &= m_1 X^{n-1} + m_2 X^{n-2} + \dots + m_k X^{n-k} + f_{n-k-1} X^{n-k-1} + \dots + f_0 \\ &= m_1 X^{n-1} + m_2 X^{n-2} + \dots + m_k X^{16} + f_{15} X^{15} + \dots + f_0 \quad (\text{for the case of } k = 15) \end{aligned} \quad (3)$$

The DLPDU presented to the PhL shall consist of an octet sequence in the specified order.

The redundant check bits $f_{n-k-1} \dots f_0$ of the FCS shall be the coefficients of the remainder $F(X)$, after division by $G(X)$, of $L(X) (X^k + 1) + M(X) X^{n-k}$

where $G(X)$ is the degree $n-k$ generator polynomial for the code words

$$G(X) = X^{n-k} + g_{n-k-1} X^{n-k-1} + \dots + 1 \quad (4)$$

and $L(X)$ is the maximal weight (all ones) polynomial of degree $n-k-1$

$$\begin{aligned} L(X) &= (X^{n-k} + 1) / (X + 1) = X^{n-k-1} + X^{n-k-2} + \dots + X + 1 \\ &= X^{15} + X^{14} + X^{13} + X^{12} + \dots + X^2 + X + 1 \quad (\text{for the case of } k = 15) \end{aligned} \quad (5)$$

That is,

$$F(X) = L(X) (X^k + 1) + M(X) X^{n-k} \text{ (modulo } G(X)) \quad (6)$$

NOTE 2 The $L(X)$ terms are included in the computation to detect initial or terminal message truncation or extension by adding a length-dependent factor to the FCS.

NOTE 3 As a typical implementation when $n-k = 16$, the initial remainder of the division is preset to all ones. The transmitted message bit stream is multiplied by X^{n-k} and divided (modulo 2) by the generator polynomial $G(X)$, specified in equation (7). The ones complement of the resulting remainder is transmitted as the $(n-k)$ -bit FCS, with the coefficient of X^{n-k-1} transmitted first.

The equation for $D(X)$ does not apply to the Type 8 protocol because that protocol sends the message and check bits in separate DLPDUs. The equation for $F(X)$ as just given does not apply to the Type 8 protocol because that protocol does not complement the check bits before transmission. The equation corresponding to $F(X)$ for the Type 8 protocol is

$$F_{\text{Type 8}}(X) = L(X) X^k + M(X) X^{n-k} \text{ (modulo } G(X)) \quad (7)$$

NOTE 4 As a typical implementation for the Type 8 protocol, the initial remainder of the division is preset to all ones. The transmitted message bit stream is multiplied by X^{n-k} and divided (modulo 2) by the generator polynomial $G(X)$, specified in equation (7). The resulting remainder without ones complementation is transmitted as the $(n-k)$ -bit FCS, with the coefficient of X^0 transmitted first.

5.3.8.2.3 At the receiving DLE

NOTE 1 This subclause does not apply to the Type 8 protocol.

The octet sequence indicated by the PhE shall be concatenated into the received DLPDU and FCS, and regarded as a vector $V(X)$ of dimension u

$$V(X) = v_1 X^{u-1} + v_2 X^{u-2} + \dots + v_{u-1} X + v_u \quad (8)$$

NOTE 2 Because of errors u can be different than n , the dimension of the transmitted code vector.

A remainder $R(X)$ shall be computed for $V(X)$, the received DLPDU and FCS, by a method similar to that used by the sending DLE (see 5.3.8.2.2) in computing $F(X)$

$$\begin{aligned} R(X) &= L(X) X^u + V(X) X^{n-k} \text{ (modulo } G(X)) \\ &= r_{n-k-1} X^{n-k-1} + \dots + r_0 \end{aligned} \quad (9)$$

Define $E(X)$ to be the error code vector of the additive (modulo-2) differences between the transmitted code vector $D(X)$ and the received vector $V(X)$ resulting from errors encountered (in the PhS provider and in bridges) between sending and receiving DLEs.

$$E(X) = D(X) + V(X) \quad (10)$$

If no error has occurred, so that $E(X) = 0$, then $R(X)$ will equal a non-zero constant remainder polynomial

$$R_{ok}(X) = L(X) X^{n-k} \text{ (modulo } G(X)) \quad (11)$$

whose value is independent of $D(X)$. Unfortunately $R(X)$ will also equal $R_{ok}(X)$ in those cases where $E(X)$ is an exact non-zero multiple of $G(X)$, in which case there are “undetectable” errors. In all other cases, $R(X)$ will not equal $R_{ok}(X)$; such DLPDUs are erroneous and shall be discarded without further analysis.

NOTE 3 As a typical implementation, the initial remainder of the division is preset to all ones. The received bit stream is multiplied by X^{n-k} and divided (modulo 2) by the generator polynomial $G(X)$, specified in equation (8).

5.3.9 Null DLPDU

A DLPDU with an empty Lpacket field, called a null DLPDU, shall be sent by a node to indicate that it has no data to send at the QoS allowed by the access opportunity.

5.3.10 Abort DLPDU

To abort a partially transmitted DLPDU, the Data Link Layer shall transmit the following sequence:

- one 0xFF octet;
- start delimiter;
- end delimiter.

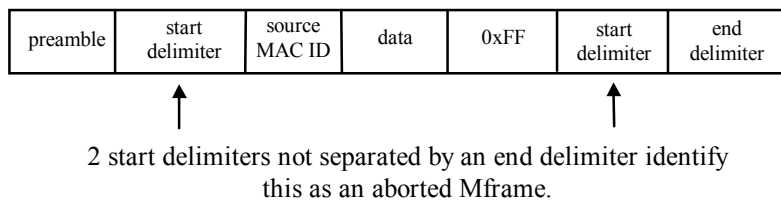


Figure 11 – Aborting a DLPDU during transmission

NOTE 1 Some implementations of this specification may build the DLPDU as it is being transmitted. If a higher layer has provided only a partial DLPDU to the Data Link Layer, and the DLL has begun transmission of the DLPDU assuming that the rest will be provided shortly, a transmit underflow occurs if the rest of the DLPDU is not provided to the DLL soon enough to complete the transmission.

NOTE 2 Although the DLL assembles the entire DLPDU including delimiters, the PhL performs start and end delimiter detection. The PhL identifies a transmit abort as a DLPDU which includes two start delimiters which are not separated by an end delimiter, as shown in Figure 11.

NOTE 3 The 0xFF octet is inserted before the start delimiter / end delimiter pair to avoid violating PhL constraints governing consecutive physical symbol relationships on the media.

5.4 Lpacket components

5.4.1 General Lpacket structure

The Lpacket (or link unit) is the PDU which is used to carry DLSDUs between peer DLS users. The components of the Lpacket shall be transmitted in the following order: size, control, tag, and link data as shown in Figure 12.

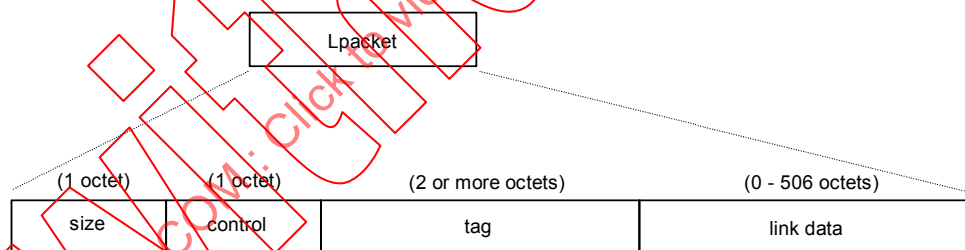


Figure 12 – Lpacket format

5.4.2 Size

The size field shall specify the number of octet pairs (from 3 to 255) contained in an individual Lpacket. The number of octet pairs shall include the size, control, tag, and link data fields counted in accordance with the following rules:

- combined, the size and control shall count as a single octet pair;
- the number of octet pairs from the tag and link data fields shall be counted separately and odd octet counts shall be rounded up;
- although the counting of octet pairs is done independently, the format of an Lpacket as placed into a DLPDU shall not include the extra octets caused by the round up.

NOTE For example, an Lpacket with a 3 octet tag (2 octet pairs) and a 9 octet link data (5 octet pairs) has a size field of $1+2+5 = 8$ octet pairs; however the Lpacket occupies $2+3+9 = 14$ octets in the DLPDU.

5.4.3 Control

5.4.3.1 Bit 0 and Bit 4 (type of Lpacket)

The bits of the control field shall be numbered 0 to 7, where bit 0 shall be the least significant bit of the control field. Bit 0 shall indicate the type of Lpacket.

- When set (bit 0 = 1), the Lpacket shall be a fixed tag Lpacket (see 4.3.4).
- When clear (bit 0 = 0), the Lpacket shall be a generic tag Lpacket (see 4.3.3).

Bit 4 of the control field shall be the inverse of bit 0. If bit 0 is clear, then bit 4 shall be set. If bit 0 is set, then bit 4 shall be clear.

5.4.3.2 Bit 1 (odd tag size)

Bit 1 of the control field shall indicate whether the tag field contains an even or odd number of octets. When clear (bit 1 = 0), this bit shall indicate that the tag contains an even number of octets. When set (bit 1 = 1), it shall indicate that the tag contains an odd number of octets.

5.4.3.3 Bit 2 (odd link data size)

Bit 2 of the control field shall indicate whether the link data contains an even or odd number of octets. When clear (bit 2 = 0), this bit shall indicate that the link data contains an even number of octets. When set (bit 2 = 1), it shall indicate that the link data contains an odd number of octets.

5.4.3.4 Bits 3, 5, 6 and 7 (reserved bits)

Bits 3, 5, 6 and 7 of the control field are reserved and shall be zero.

5.4.4 Generic tag Lpackets

Generic link-units are formed from the DLS-generic-tag and DLS-user-data provided by the DLS user with the addition of size and control field information for generic tags as shown in Figure 13. The 3 octet generic tag address (see 4.3.3) shall identify the connected mode information carried in the link data field.

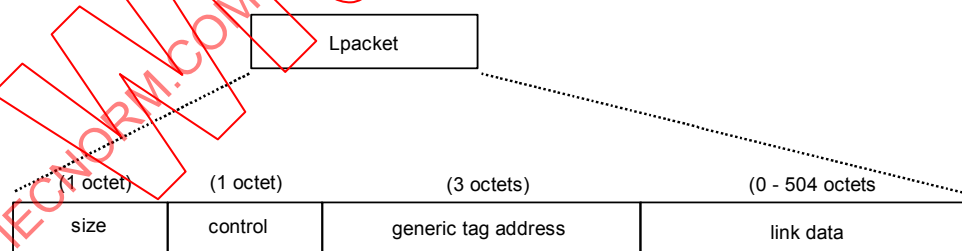


Figure 13 – Generic tag Lpacket format

The minimum length generic tag Lpacket shall be 5 octets (minimum link data size of 0 octets). The maximum size of link data shall be 504 octets.

NOTE Generic Lpackets are usually connection mode transfers with the QoS value of scheduled or low.

5.4.5 Fixed tag Lpackets

Fixed tag link-units are formed from the DLS-fixed-tag and DLS-user-data provided by the DLS user with the addition of size and control field information for fixed tags as shown in Figure 14. The 2 octet fixed tag address (see 4.3.4) shall identify the service access point in the the destination node and the address of the destination node.

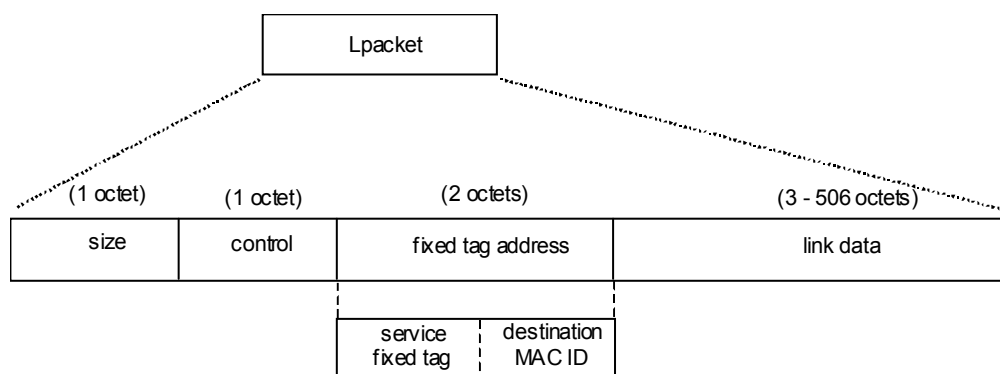


Figure 14 – Fixed tag Lpacket format

The minimum length fixed tag Lpacket shall be 7 octets (minimum link data size of 3 octets). The maximum size of link data shall be 506 octets.

NOTE Fixed Lpackets are always connectionless mode transfers. They are not usually sent with the QoS parameter value scheduled.

5.5 DLPDU procedures

5.5.1 General

The transmission protocol combines one or more Lpackets into single DLPDUs for sending according to the available transmission opportunity. The DLPDU components are passed to the Ph layer by the transmit machine framer (TxM) and Serializer (see 9.3).

A node shall always send one DLPDU at each valid access opportunity for it to transmit, and no DLPDU shall include Lpackets with a lesser QoS parameter than the QoS value applicable to the access opportunity.

If a node has no Lpackets available at or better than the QoS of the access opportunity, then a Null DLPDU shall be sent.

If for internal reasons a node begins a DLPDU and is unable to complete it, the abort sequence shall be sent, transmission shall stop for that access opportunity and the local DLS user shall be notified.

NOTE Null DLPDUs and abort DLPDUs provide an efficiency improvement as they notify other nodes that use of the transmission slot has ended and enable the next node to begin transmission without waiting for slot time out.

Each Lpacket in the queue awaiting transfer has an associated local identifier. At completion of the transfer to the PhL, or an aborted transfer, the local identifier is returned to the local DLS user with a status confirmation of OK or TXABORT.

The DLS user has additional local services to identify unsent Lpackets and request they are flushed from the queue, this action results in the confirmation status of FLUSHED being returned with the local identifier for Lpackets that have been flushed.

5.5.2 Sending scheduled DLPDUs

Scheduled DLPDUs are normally used for transfer of connected mode operation using Generic Lpackets. They are formed by concatenating multiple Lpackets (subject to a limit of 510 octets), prepending Ph-control information and source station MAC ID, and appending a defined frame check sequence and further Ph-control information.

Lpackets are packed into a Scheduled DLPDU in FIFO order as received from the DLS user, until the queue is empty of packets with Scheduled QoS or the next such Lpacket size exceeds the remaining space.

5.5.3 Sending unscheduled DLPDUs

Unscheduled DLPDUs may contain both Generic Lpackets and Fixed Lpackets. They are formed by concatenating multiple Lpackets (subject to a limit of 510 octets), prepending Ph-control information and source station MAC ID, and appending a defined frame check sequence and further Ph-control information.

Lpackets are packed into an unscheduled DLPDU in QoS priority order (Scheduled, High and Low priorities respectively). Within each QoS, the FIFO order of queuing by the DLS user is followed, until the queue is empty or the next such Lpacket size exceeds the remaining space.

5.5.4 Receiving DLPDUs

5.5.4.1 Unpacking

Received DLPDUs are assembled from the incoming Ph stream of symbols and unpacked by the Octet Deserializer, Receive Machine Frammer (RxM) and Receive Logical Link Control (RxLLC) functions in two stages

- a frame check sequence is computed over the stream of received octets, (excluding Ph delimiters) and checked for the proper residual value, if this is correct, then
- each Lpacket contained in the DLPDU is examined to extract its tag. Each extracted tag is then compared with the contents of a local tag filter to determine if the associated Lpacket is of interest to the receiving station. Lpackets whose tags match the tag filter are then processed further and passed to the DL user, DL management or used inside the DLL. Indications passed to the DL user or DL management include Lpacket size, Lpacket tag and Lpacket data. In addition for fixed tag Lpackets, the source ID is also indicated.

5.5.4.2 Tag filtering

Each local DLS provider maintains a list of tags used to identify Lpackets of interest to its DL user or its internal DL management functions. The DL user may access and change this list to specify

- generic tags of interest;
- fixed tags of interest.

NOTE A generic tag specifies the Lpackets for a single connection-mode object. A fixed tag specifies all connectionless-mode messages in a particular class, for example Time distribution.

5.6 Summary of DLL support services and objects

The operation and coordination of participating DLL providers is governed by variables and attributes held in DL management objects or provided by the DL user services listed in Table 8. Access to these items is via defined fixed tag addresses, DL user services or unspecified local services. Details are given in the listed clauses.

Table 8 – DLL support services and objects

DLL item	Description	Location
DLS user services		see IEC 61158-3-2
connected mode transfer service	a local service for requesting, indicating and confirming generic tag (connected mode) data transfer by Lpackets	see 6.2.2
connectionless mode transfer service	a local service for requesting, indicating and confirming generic tag (connected mode) data transfer by Lpackets	see 6.2.2
queue maintenance service	a local service for flushing unsent Lpacket messages from the internal DL service queue	see 6.2.3
tag filter service	a local service for the DLS user to inform its local DL provider which Lpacket tags and tag types are to be accepted and processed by the DLL or passed to the DL user	see 6.2.4
link synchronization service	a local service for informing the local DLS user of the current NUT number to maintain schedule pointers	see 6.2.5
synchronized parameter change service and tMinus-start-countdown	a local service for loading and changing local lists of current and pending DL configuration data and in conjunction with the tMinus-start-countdown service and moderator DLPDU, to manage an accurately timed change over to new DL link configurations	see 6.2.6
event reports service	local services to report various events for use by management	see 6.2.7
bad FCS service		see 6.2.8
current moderator		see 6.2.9
power-up and on-line		see 6.2.10
enable moderator		see 6.2.11
listen only		see 6.2.12
Generic tag service	an Lpacket service for connected mode communication	see 6.3
Fixed tag services	a range of Lpacket service addresses for various connectionless mode communication	see 5.4.5
moderator	an Lpacket broadcast in every NUT for managing time precision and timing for change of link parameters	see 6.4
time distribution service	an Lpacket for publishing time values and offsets used in coordinating the same time sense in all stations	see 6.5
UCMM	an Lpacket service to transfer DLSDUs from a DLS user to the Unconnected Message Manager in a peer DLS-user	see 6.6
Keeper UCMM	an Lpacket service to transfer DLSDUs from a DLS user to the Keeper objects in other nodes	see 6.7
TUI	an Lpacket broadcast to all stations assisting their synchronization to the current link configuration	see 6.8
link parameters	an Lpacket service for publishing pending link parameters in advance of a synchronized change of link operating conditions	see 6.9
tMinus	an Lpacket service to start a count down leading to a change of link parameters and return the result.	see 6.9
I'm alive	an Lpacket broadcast by new and reset devices to assist their entry into an operating system	see 6.10
ping request and reply	an Lpacket service soliciting identification responses from all other active devices on the link	see 6.11
WAMI	an Lpacket allowing temporary devices to identify a local device MAC ID when they are plugged into its network access port	see 6.12
debug	an Lpacket service to trace object state transitions	see 6.13

DLL item	Description	Location
Objects		
ControlNet object	holds and distributes station management data classes and instances for Ph and DL layers, also collects diagnostic information for use by other DLS-user entities	see 7.2
Keeper object	accessed via the Keeper UCMM, or general UCMM fixed tag, to provide a range of support services for link operation, schedules and connections holds link data and attributes for all devices on a link keeps copies of all connection originator data for connection originating devices in a network negotiates with other Keeper objects on a link to agree on master Keeper and backup Keeper roles when master Keeper distributes link attributes to all nodes and keeps backup Keepers synchronized regularly issues TUI Lpacket to ensure all stations are synchronized to the current links configuration manages the network resource semaphore enabling orderly modification of link attribute data	see 7.3
Scheduling object	provides services to external link scheduling software in support of read/write for connection and schedule information from involved devices stores the current schedule details and connection password in the Keeper	see 7.4
TCP/IP Interface object	provides the mechanism to configure the TCP/IP interface of a device	see 7.5
Ethernet Link object	maintains link-specific counters and status information for a physical Ethernet ISO/IEC 8802-3 port	see 7.6
DeviceNet object	holds and distributes station management data classes and instances for Ph and DL layers, also collects diagnostic information for use by other DLS-user entities	see 7.7
Connection Configuration object	provides an interface to create, configure and control connections in a device	see 7.8

6 Specific DLPDU structure, encoding and procedures

6.1 Modeling language

6.1.1 State machine description

Some of the components of the Data Link Layer are specified using a state machine description language. The action statement is expressed in C++. Other components do not require a state machine description and are specified using only C++.

A state machine is described using the following syntax, where * indicates repetition of one or more of the preceding component, and {} indicates an optional component:

```

state-machine :=
    <header-statements>
    state*
state :=
    "state:" <state-name>
    transition*
transition :=
    "event:" <event-expression>
    { "condition:" <condition-expression> }
    "destination:" <state-name>
    { "action:" <action-statements> }

```

State-names are legal C++ identifiers, event-expressions and condition-expressions are C++ expressions, and header-statements and action-statements are lists of C++ statements.

Header-statements include

- #include and #define;
- variable and interface declarations;
- subroutine definitions.

An event-expression indicates the specified event when the value of the expression is TRUE. An event is required to trigger a transition. A condition-expression qualifies a transition. A qualified transition occurs when its event occurs if the condition is true at the time of the event. In general, event-expressions are FALSE upon entry into a state. There are a few cases where an event-expression may be true upon entry to a state. In these cases, the transition takes place immediately.

Action statements are executed when the transition is triggered.

All the event triggers and conditions of the current state are evaluated continuously and concurrently until a given transition is enabled. All expressions and statements execute instantaneously. The following model illustrates the precedence of the various parts of a transition:

```
if(the event has occurred && the condition is true)
{
  do the actions;
  go to the destination;
}
```

Implementations may use other code or syntax, however the implementation performance shall be equivalent to that specified except for internal execution timing and local variables

6.1.2 Use of DLL- prefix

The primitives and parameters described in IEC 61158-3-2 use the following prefixes:

- DL- for data link
- DLS- for DL-link service
- DLMS- for DL-management service.

Within this Type 2: Data Link Protocol Specification, the generic prefix DLL- is used as equivalent to the appropriate service description prefix.

6.1.3 Data types

The data types used in the data link variables, parameters, state machines and example code are specified in IEC 61158-5-2.

The specification of data types corresponds to the notation of IEC 61131-3. A list of the elementary data types, their corresponding description and some of their valid values is shown in Table 9. Additional detail is specified in IEC 61158-6-2.

Table 9 – Elementary data types

Keyword	Description	Range	
		Minimum	Maximum
BOOL	Boolean		
SINT	Short Integer	-128	127
INT	Integer	-32 768	32 767
DINT	Double Integer	-2 ³¹	2 ³¹ -1
LINT	Long Integer	-2 ⁶³	2 ⁶³ -1
USINT	Unsigned Short Integer	0	255
UINT	Unsigned Integer	0	65 535
UDINT	Unsigned Double Integer	0	2 ³² -1
ULINT	Unsigned Long Integer	0	2 ⁶⁴ -1
REAL	Floating point		
LREAL	Long float		
ITIME	Duration (short)		
TIME	Duration		
FTIME	Duration (high resolution)		
LTIME	Duration (long)		
DATE	Date only		
TIME_OF_DAY or TOD	Time of day		
DATE_AND_TIME or DT	Date and time of day		
STRING	character string (1 octet per character)		
STRING2	character string (2 octet per character)		
STRINGN	character string (N octets per character)		
SHORT_STRING	character string (1 octet per character, 1 octet length indicator)		
SWORD	bit string - 8 bits		
WORD	bit string - 16-bits		
DWORD	bit string - 32-bits		
LWORD	bit string - 64-bits		

6.2 DLS user services

6.2.1 General

The DLL user services provided to the DLS user have the following interfaces used for the specified purposes. The related service definitions and time sequence diagrams are given in IEC 61158-3-2.

6.2.2 Connected mode and connectionless mode transfer service

6.2.2.1 Service encoding

The request and confirmation services used to queue the sending of Lpackets by the DLL shall be of the following forms.

- Connection mode transfers use Generic tag request and confirmation.

```

DLL_xmit_generic_request(
    IDENTIFIER id,
    USINT Lpacket[],
    UINT Lpacket_size,
    PRIORITY priority,
    USINT gentag[3] );

DLL_xmit_generic_confirm(
    IDENTIFIER id,
    TXSTATUS status );

```

b) Connectionless mode transfers use Fixed tag request and confirmation.

```

DLL_xmit_fixed_request(
    IDENTIFIER id,
    USINT Lpacket[],
    UINT Lpacket_size,
    PRIORITY priority,
    USINT service,
    USINT destID );

DLL_xmit_fixed_confirm(
    IDENTIFIER id,
    TXSTATUS status );

```

c) Connection mode transfers use Generic tag indication.

```

DLL_recv_generic_indication(
    USINT Lpacket[],
    UINT Lpacket_size,
    USINT gentag[3] );

```

d) Connectionless mode transfers use Fixed tag indication.

```

DLL_recv_fixed_indication(
    USINT Lpacket[],
    UINT Lpacket_size,
    USINT service,
    USINT sourceID );

```

6.2.2.2 Procedures for request and confirmation

The `Lpacket` parameter shall contain the DLSDU as an ordered sequence of octets. The `Lpacket_size` parameter shall specify the number of octets contained in the `Lpacket` array. The QoS priority shall specify onto which internal queue the request shall be placed and shall be one of LOW, HIGH or SCHEDULED. Only Lpackets on the SCHEDULED queue may be sent during the scheduled portion of the NUT. The unscheduled portion of the NUT shall be used to transmit Lpackets from the HIGH and LOW QoS priority queues, and may be used to transmit Lpackets from the SCHEDULED queue. The queues shall be prioritized such that, during an unscheduled transmit opportunity, the SCHEDULED queue is emptied first, then the HIGH QoS priority queue is emptied second, and finally the LOW QoS priority queue is emptied last.

The `IDENTIFIER id`, shall correlate a confirmation with a corresponding request. The `status` parameter shall return the status of the attempted transmit and shall be one of OK, TXABORT or FLUSHED.

The `DLL_xmit_fixed_request` service queues an Lpacket with a two octet fixed tag for transmit. The `service` parameter shall specify the first octet of the fixed tag Lpacket. The `destID` parameter shall specify the second octet, which shall determine the MAC ID of the destination node.

The remaining parameter, `gentag`, of the `DLL_xmit_generic_request` service shall specify on which generic tag to transmit the Lpacket.

NOTE 1 The use of one or more queues for different priorities is an internal implementation choice.

NOTE 2 Assembly of the related Lpacket structure and the procedure for its sending on the link are described in their respective clauses (see 5.3 and 5.4).

6.2.2.3 Procedures for indication

These indications shall signal the delivery of an Lpacket with a good FCS. Only fixed tag Lpackets which have had their service enabled by a successful call to `DLL_enable_fixed_request` shall cause a `DLL_recv_fixed_indication`. Only generic tag Lpackets which have had their generic tag enabled by a successful call to `DLL_enable_generic_request` shall cause a `DLL_recv_generic_indication`.

The array of octets, `Lpacket`, shall contain the DLSDU as an ordered sequence of octets. The integer, `Lpacket_size`, shall specify the number of octets contained in the `Lpacket` array.

6.2.3 Queue maintenance service

6.2.3.1 Service encoding

Once queued, an `Lpacket` shall remain in a transmit queue until it is transmitted or de-queued. The request and confirmation services used to de-queue an `Lpacket`, before it is transmitted, shall be of the form:

```
DLL_flush-requests-by-QoS_request( PRIORITY priority );
DLL_flush-requests-by-QoS_confirm( PRIORITY priority );
DLL_flush_single_request( PRIORITY priority, IDENTIFIER id );
```

6.2.3.2 Procedures for request and confirmation

`DLL_flush-requests-by-QoS_request` shall cancel all pending transmits at the specified QoS priority. Each pending transmit shall immediately terminate with a confirmation containing the `FLUSHED` status. `DLL_flush_single_request` shall cancel a specific `Lpacket` identified by the `id` used to transmit the `Lpacket`. This service need not have a corresponding confirmation since the confirmation of the queued `Lpacket` shall serve this purpose.

6.2.4 Tag filter service

6.2.4.1 Service encoding

By default, the Data Link Layer shall only process the moderator `Lpacket` (fixed tag 0x00), and all other `Lpackets` shall be discarded. Other protocol layers may enable or disable reception of specific `Lpackets` using the following services.

```
DLL_enable_generic_request( IDENTIFIER id, USINT gentag[3] );
DLL_disable_generic_request( IDENTIFIER id, USINT gentag[3] );
DLL_enable_fixed_request( IDENTIFIER id, USINT service );
DLL_disable_fixed_request( IDENTIFIER id, USINT service );
DLL_enable_generic_confirm( IDENTIFIER id, BOOL result );
DLL_disable_generic_confirm( IDENTIFIER id, BOOL result );
DLL_enable_fixed_confirm( IDENTIFIER id, BOOL result );
DLL_disable_fixed_confirm( IDENTIFIER id, BOOL result );
```

6.2.4.2 Procedures for request and confirmation

The `IDENTIFIER`, `id`, shall correlate a confirmation with a corresponding request. The parameter, `result`, shall return `TRUE` if the request was successful and `FALSE` if unsuccessful. The `gentag` parameter shall specify which generic tag to route to the higher layer that enabled the tag; likewise, the `service` parameter shall specify which fixed tag to route to the higher layer that enabled the tag.

NOTE The `DLL_enable_generic_request` service can return an unsuccessful result if the implementation of the Data Link Layer is unable to filter any more generic tags.

6.2.5 Link synchronization service

6.2.5.1 Service encoding

The indication that a new network update time (NUT) has begun shall be of the form:


```
DLL_tone_indication( USINT cycle );
```

6.2.5.2 Procedures for indication

The parameter `cycle` shall return the interval counter from the moderator DLPDU just received.

NOTE If a moderator DLPDU is expected but does not arrive, the local node simulates the reception of the moderator DLPDU based on an internal timer of the ACM.

6.2.6 Synchronized parameter change service

6.2.6.1 Service encoding

All nodes on a link shall maintain two copies of link parameters: current and pending. The current copy of link parameters shall be used for the on going operation of the Data Link Layer. The pending copy shall be maintained to allow a synchronized change of link parameters. The local DLL user services that read and write these link parameters shall be of the form:

```
DLL_set_pending_request(          DLL_set_pending_confirm(
    IDENTIFIER id,                IDENTIFIER id,
    DLL_config_data params );      BOOL result );

DLL_set_current_request(          DLL_set_current_confirm(
    IDENTIFIER id,                IDENTIFIER id,
    DLL_config_data params );      BOOL result );

DLL_get_pending_request(          DLL_get_pending_confirm(
    IDENTIFIER id );              IDENTIFIER id,
                                   DLL_config_data params );

DLL_get_current_request(          DLL_get_current_confirm(
    IDENTIFIER id );              IDENTIFIER id,
                                   DLL_config_data params );
```

The `DLL_config_data` shall contain the link parameters and be of the form:

```
class DLL_config_data
{
public:
    USINT myaddr; // the MAC ID of this node
    UINT NUT_length; // the length of the NUT in 10 µs increments
    USINT smax; // highest MAC ID allowed to transmit scheduled
    USINT umax; // highest MAC ID allowed to transmit unscheduled
    USINT slotTime; // time allowed for line turnaround in 1 µs increments
    USINT blanking; // time to disable RX after DLPDU in 1,6 µs increments
    USINT gb_start; // 10 µs intervals from start of guardband to tone
    USINT gb_center; // 10 µs intervals from start of moderator to tone
    USINT modulus; // modulus of the interval counter
    USINT gb_prestart; // transmit cut-off, 10 µs intervals before tone
}; // may not transmit passed this limit
```

The local DLL user services which request and confirm the start of a `tMinus` countdown shall be of the form:

```
DLL-tMinus-start-countdown-request(          DLL-tMinus-start-countdown-confirm(
    IDENTIFIER id,                            IDENTIFIER id,
    USINT start_count );                      BOOL result );
```

NOTE The `tMinus` fixed tag service (see 6.9) is used by the master Keeper to request all stations on the link to initiate a changeover countdown. This is the local DLL user procedure to request and confirm local participation.

At the correct expiry of the `tMinus` countdown, each participating station passes a local indication to its ControlNet object requesting a change from current link parameters to pending link parameters.

```
DLL_tminus_zero_indication( );
```

6.2.6.2 Procedures for request, confirmation and indication

The IDENTIFIER, *id*, shall correlate a confirmation with a corresponding request. The *result* shall return TRUE if the request was successful and FALSE if unsuccessful.

The moderator Lpacket contains a field, called *tMinus*, that shall be used to synchronize the update of the current link parameters. The *DLL-tMinus-start-countdown-request* shall cause a node to participate in a *tMinus* countdown, and, if the node is the moderator, shall initialize the *tMinus* field of the moderator Lpacket. The moderator node shall decrement this field before transmitting each moderator until the field equals zero. When the *tMinus* field transitions from 1 to 0, the DLL in each node participating in the countdown shall locally generate a *DLL_tminus_zero_indication* and copy its pending link parameters into its current copy. If the *tMinus* field transitions to 0 from any value except 1, the countdown shall be aborted and no *DLL_tminus_zero_indication* shall be generated.

6.2.7 Event reports service

The indication used to alert the Station Management entity of various events internal to the Data Link Layer shall be of the form:

```
DLL_event_indication(
    DLL_event event,
    USINT mac_id /*[optional]*/ );
```

The *DLL_event*, *event*, shall be one of the events enumerated in Table 10. The *mac_id* parameter shall be used in the case of (*event* = *DLL_EV_badFrame*); it shall indicate the source MAC ID of the node that transmitted the bad DLPDU.

NOTE Since the DLPDU was damaged, the source MAC ID may not be correct.

Table 10 – DLL events

DLL events	Description
DLL_EV_rxGoodFrame	A good DLPDU was received. This shall include DLPDUs that contain no data (null DLPDUs), but shall exclude moderators
DLL_EV_txGoodFrame	A good DLPDU was transmitted. This shall include DLPDUs that contain no data (null DLPDUs), but shall exclude moderators
DLL_EV_badFrame	A damaged DLPDU was received. The optional parameter, which is the apparent source MAC ID of the offending node, shall also be reported
DLL_EV_errA	A bad DLPDU was received on channel A of the physical medium, or a good DLPDU was received on channel B and <code>ph_frame_indication</code> from channel A stayed false
DLL_EV_errB	A bad DLPDU was received on channel B of the physical medium, or a good DLPDU was received on channel A and <code>ph_frame_indication</code> from channel B stayed false
DLL_EV_txAbort	A transmit DLPDU was terminated with an abort sequence
DLL_EV_NUT_overrun	NUT is not large enough to accommodate all the scheduled traffic
DLL_EV_dribble	Scheduled Lpackets could not be sent during scheduled time
DLL_EV_nonconcurrency	An event was detected that indicates that this node is out of step with the access control protocol
DLL_EV_rxAbort	A DLPDU was received that was terminated with an abort sequence
DLL_EV_lonely	Have not heard a DLPDU from another node on the link for 8 NUTs
DLL_EV_dupNode	Another node on the link is using this node's MAC ID
DLL_EV_noisePulse	<code>ph_lock_indication</code> went true then false before <code>ph_frame_indication</code> went true, but <code>ph_lock_indication</code> was not true long enough to indicate a possibly damaged DLPDU
DLL_EV_collision	<code>ph_frame_indication</code> was true when this node was about to transmit
DLL_EV_invalidModAddress	A moderator was received from a node that does not have the lowest MAC ID on the link
DLL_EV_rogue	A moderator DLPDU was received that does not match the link configuration information at this node
DLL_EV_deafness	Cannot hear the moderator DLPDU even though other link traffic is present
DLL_EV_supernode	A moderator was received from MAC ID 0

6.2.8 Bad FCS service

The indication used to alert the Station Management entity that a received DLPDU had an invalid FCS shall be of the form:

```
DLL_badFCS_indication( CHANNEL channel );
```

The `channel` parameter shall indicate which PhL entity provided the DLPDU which was in error. This parameter shall be one of CHA or CHB, corresponding to PhL channel A and PhL channel B, respectively. This indication shall be provided, at most, once per error DLPDU per channel.

6.2.9 Current moderator service

The indication used to inform the Station Management entity which node is currently the moderator shall be of the form:

```
DLL_currentMod_indication( USINT macID );
```

The `macID` parameter shall indicate the MAC ID of the node that last transmitted a valid moderator DLPDU.

6.2.10 Power up and online services

6.2.10.1 Service encoding

The request and confirmation which places the Data Link Layer on-line shall be of the form:

```
DLL_online_request( BOOL online );           DLL_online_confirm( BOOL online );
```

The indication that specifies that the Data Link Layer has completed its initialization shall be of the form:

```
DLL_powerup_indication( void );
```

6.2.10.2 Procedures for request and confirmation

At power-up, the DLL shall wait until the `online` parameter equals TRUE. The DLL shall then begin the process of going on-line. When the `online` parameter is FALSE, the DLL shall go off-line at the end of the current NUT. When off-line the DLL shall not transmit, and shall ignore any link activity.

6.2.11 Enable moderator service

6.2.11.1 Service encoding

The request and confirmation services that enable and disable the ability of a node to assume the moderator role shall be of the form:

```
DLL_enable_moderator_request( BOOL enable );
DLL_enable_moderator_confirm( BOOL enable );
```

6.2.11.2 Procedures for request and confirmation

When the `enable` parameter is TRUE, the DLL shall enable the transmission of moderator DLPDUs. When the `enable` parameter is FALSE, transmission of moderator DLPDUs shall be disabled.

NOTE This request is used by station management entities to allow a new node to non-disruptively join a working link. The moderator switch over protocol does not tolerate the node with the lowest MAC ID suppressing moderator DLPDUs for an extended period. The Network Attachment Monitor (NAM) re-enables moderator transmission whenever another device is detected on the link.

6.2.12 Listen only service

6.2.12.1 Service encoding

The request and confirmation services that allows a device to receive, but disables the ability of a device to transmit, shall be of the form:

```
DLL_listen_only_request( BOOL enable );
DLL_listen_only_confirm( BOOL enable );
```

6.2.12.2 Procedures for request, confirmation and indication

When the `enable` parameter is TRUE, the Data Link Layer shall participate in the access protocol and transmit DLPDUs. When the `enable` parameter is FALSE, transmission of DLPDUs shall be disabled; however, the ability of the node to receive DLPDUs shall not be impaired.

6.3 Generic tag Lpacket

6.3.1 General

The generic tag Lpacket is used for transferring connection data. The connection ID or Generic Tag is a unique identifier for previously negotiated resources and parameters at its

source, destination(s) and any intermediate transit points. Only the generic Tag is necessary to identify the message data.

6.3.2 Structure of the generic-tag Lpacket

The size and control fields shall follow the Generic Tag format. The address shall be the DLS-generic-tag provided by the local DL-generic-request service. The link data field shall contain the DLS-user-data provided by the DL-generic-request.

6.3.3 Sending and receiving the generic-tag Lpacket

The sending station shall send the Lpacket at the QoS priority specified by the DL-generic-request.

The status of each generic tag Lpacket transmission is confirmed to the DL user by returning the DLS-user-id (an internal handle or identifier) and the DLS Txstatus with one of the following values.

- a) "OK" — message successfully sent;
- b) "TxABORT" — sending process failed;
- c) "FLUSHED" — message has been removed from the pending queue before sending.

NOTE 1 Txstatus is a local variable so the coding is not specified.

NOTE 2 The FLUSHED status is only used in response to a deletion requested by the Queue maintenance service

NOTE 3 The parameter value OK is not an indication that the message has been received

All receiving stations whose local Tag Filter service contains an address matching the received Lpacket address, shall pass each correctly received data unit to its local DL user, together with the data unit size and the address (the generic tag for the connection).

6.4 Moderator Lpacket

6.4.1 General

The moderator MAC Lpacket is used for DLL management, access timing and synchronization, it is sent as a moderator DLPDU containing only one Lpacket, the moderator Lpacket.

6.4.2 Structure of the moderator Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the moderator fixed tag and 0xFF the broadcast address for all MAC IC nodes on this link. The link data field shall contain the following in the listed order.

```
class moderatorLpacket : public fixedLpacket
{
public:
    UINT    NUT_length;        // the length of the NUT in 10 µs increments
    USINT   smax;              // highest MAC ID allowed to transmit scheduled
    USINT   umax;              // highest MAC ID allowed to transmit unscheduled
    USINT   slotTime;          // 1 µs increments allowed for line turnaround
    USINT   blanking;          // 1,6 µs increments to disable RX after DLPDU
    USINT   gb_start;          // 10 µs intervals from start of guardband to tone
    USINT   gb_center;         // 10 µs intervals from start of moderator to tone
    USINT   usr;               // unscheduled start register
    USINT   interval_count;    // current NUT number (0 to modulus)
    USINT   modulus;           // modulus of the interval counter
    USINT   tMinus;            // countdown to synchronized link parameter change
    USINT   gb_prestart;       // transmit cut-off, 10 µs intervals before tone
    USINT   reserved;         //
};
```

6.4.3 Sending and receiving the moderator Lpacket

The moderator DLPDU containing the moderator Lpacket shall be transmitted once per NUT by the node with the lowest MAC ID. It shall be sent in the guardband, a fixed part of every NUT reserved for the moderator DLPDU.

To support DLL management actions, the sender and receivers of the moderator DLPDU shall use the contents of the moderator link data field as follows.

The `NUT_length` shall specify the duration of the current NUT. `NUT_length` shall be a 16 bit integer representing the number of 10 μ s ticks and shall be in the range 50 (500 μ s) to 10 000 (100 ms). The beginning of the NUT shall be called the tone. At the tone, a counter that decrements every 10 μ s shall be loaded with the `NUT_length`.

NOTE 1 Subclause 8.2 further constrains the value of `NUT_length`.

To guarantee that the link is quiet during moderator transmission, all nodes shall cease transmitting when their internal counter reaches the value of `gb_prestart`. All nodes may then assume that the link is quiet between `gb_start` and `gb_center`. When the moderator node's counter reaches the value of `gb_center`, it shall begin transmitting a DLPDU containing only the moderator Lpacket. The moderator Lpacket shall be completely transmitted at least 30 μ s before the decrementing counter would have expired.

NOTE 2 Subclause 8.2 calculates the values of `gb_prestart`, `gb_start`, and `gb_center` to guarantee these constraints. Factors that affect the calculation of these values include clock accuracy, missing nodes, and propagation delay between nodes. In all cases, `gb_prestart` > `gb_start` > `gb_center` > 7.

The highest MAC ID transmitting during the scheduled portion of the NUT shall be `smax`. It shall be in the range 0 to 254. `umax` shall determine the highest MAC ID that transmits during the unscheduled portion of the NUT. It shall be in the range `smax` to 254. The unscheduled start register, `usr`, shall select which node transmits first in the unscheduled portion of the NUT. The `usr` shall increment each NUT. This counter shall be updated modulo (`umax` + 1).

A node shall listen to the PhL directly after it finishes transmitting. `blanking` shall determine the amount of time to suspend listening and shall be in 1,6 μ s ticks.

The moderator node shall increment an 8 bit counter, called `interval_count`, once per NUT. This counter shall be updated modulo (`modulus` + 1). Since the guardband is at the end of the NUT, the value of `interval_count` shall correspond to the NUT that just completed.

NOTE 3 Subclause 8.2 and the specifications of the ControlNet object further constrain the value of `smax`, `umax`, `blanking` and `modulus`.

The `tMinus` field shall maintain a countdown until all nodes adopt new link parameters (see 6.9). The `slotTime` field shall determine the time to wait for a missing node and shall be in 1,0 μ s ticks. An additional 1,0 μ s shall be added to the value of the `slotTime`. The `slotTime` field shall be in the range 8 to 254; the other values shall be reserved. The reserved octet shall be set to zero.

6.5 Time distribution Lpacket

6.5.1 General

The moderator DLPDU provides a common reference marker that is synchronized between all nodes on the network. By distributing and processing time stamps relative to the reference instead of to the time distribution message, implementations are simplified whilst accuracy is improved by several orders of magnitude. Phase and frequency synchronization is inherent in this DL-protocol to a very high level of accuracy. The accuracy of time synchronization using the time distribution format defined in this subclause is implementation dependent, however it can be better than 10 μ s.

6.5.2 Structure of the time distribution Lpacket

6.5.2.1 Packet format

The size and control fields shall follow the fixed tag format. The address shall be formed from the time distribution fixed tag and 0xFF the broadcast address. The link data field shall contain the following in the listed order:

```
class TimeDist_Lpacket : public fixedLpacket
{
public:
    USINT    revision;    // revision of time distribution format
    USINT    leap;        // leap second offset
    UINT     goodness;    // time relay control field
    DINT     gse;         // global squared error relative to ultimate master
    LINT     dctz;        // distribution channel time zero
    LINT     ts_ref;      // time stamp of previous reference pulse
    LINT     ts_tx;       // time stamp of this message's transmission
};
```

6.5.2.2 Revision

The revision parameter shall be set to zero. This parameter shall represent the revision of the time distribution format.

6.5.2.3 Leap

The leap parameter shall specify the Universal Coordinated Time (UTC) leap seconds. This number, when added to the system time, shall give actual UTC time. This number can increment or decrement by one twice a year on the solstices, as dictated by the US Naval Observatory, to maintain synchronism between terrestrial and astronomical time. If zero, then the number of leap seconds is unknown.

NOTE The leap parameter should not be used in any control situations, but may be needed in some time relays to distribution channels that are based on UTC rather than Global Positioning Satellite (GPS) time.

6.5.2.4 Goodness

6.5.2.4.1 Parameter structure

The goodness parameter shall be partitioned as shown in Figure 15.

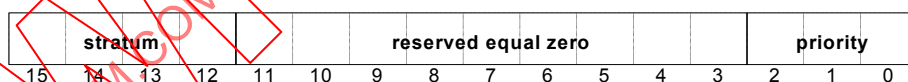


Figure 15 – Goodness parameter of TimeDist_Lpacket

6.5.2.4.2 Stratum

The most significant field, called stratum, shall specify the number of time relays between this message, and a source of absolute time. A value of 0 shall signify an exact reference, and the value shall be incremented for every intermediate time relay. If the priority field is set to zero (lock not achieved), or the number of intermediate time relays exceeds 15, the goodness parameter shall be set to 15. Bits 3 through 11 shall be reserved and set to zero.

NOTE A time relay is a link-to-link router which distributes time synchronization messages on its links based on the time synchronization messages received on its other links.

6.5.2.4.3 Source quality

The priority field shall indicate the quality of the time message as shown in Table 11.

Table 11 – Time distribution priority

Value	Meaning
7	Absolute system time. Acting as master
6	Absolute system time. Acting as dependent
5	Human set system time. Acting as master
4	Human set system time. Acting as dependent
3	Lock established to node on distribution channel other than this one
2	Lock established to node on this channel. System time unknown
1	(invalid)
0	Not synchronized with any other node

6.5.2.5 gse

The gse parameter shall indicate the cumulative rms stability squared. This parameter shall approximate the node's worst case stability relative to the rest of the system. The units of this parameter shall be $(0,1 \mu\text{s})^2$. When the rms stability is unknown or not yet determined, it shall be 0xFFFFFFFF.

6.5.2.6 LINT time parameters

In each of the LINT time parameters, the most significant bit shall be zero – only 63 bits shall be used. The units of these parameters shall be $0,1 \mu\text{s}$.

6.5.2.7 dctz

The dctz parameter shall indicate the system time offset from the distribution channel's arbitrary time zero, established when the networks are synchronized.

6.5.2.8 ts_ref

The ts_ref parameter shall indicate the time stamp of the last tone following a moderator DLPDU which had its interval_count equal to zero. The value of zero shall indicate that this value is not known. System time zero shall be defined as that used for the Global Positioning Satellites: 12:00 midnight, Jan 6, 1980 GMT.

6.5.2.9 ts_tx

The ts_tx parameter shall indicate the time stamp at the transmission of this message. The value of zero shall indicate that this value is not known. System time zero shall be defined as that used for the Global Positioning Satellites: 12:00 midnight, Jan 6, 1980 GMT. This parameter shall be set to zero.

NOTE This protocol does not use GPS time, it only uses the time zero point for GPS. So the 1999 roll over of GPS had no relevance for system time.

6.5.3 Sending and receiving the time distribution Lpacket

The current master Keeper for the link is responsible to sent the time distribution Lpacket at intervals and QoS values configured by the DL user to match its application requirements.

All nodes are responsible to receive the time distribution Lpacket.

6.6 UCMM Lpacket

6.6.1 General

The Unconnected Message Manager (UCMM) is a DLS-user entity providing an unconnected request/response message service for at least one outstanding message. It uses the DLL transmit service and the UCMM Lpacket for its messages.

6.6.2 Structure of the UCMM Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the UCMM fixed tag and destination MAC ID provided by the DLL transmit service. The link data field shall contain the DLS user data provided by the DLL transmit service.

6.6.3 Sending and receiving the UCMM Lpacket

The UCMM Lpacket shall be sent at the QoS priority specified by the DLL data transfer service.

The status of each UCMM transmission is confirmed to the DL user by returning the DLS-user-id (an internal handle or identifier) and the DLS Txstatus with one of the following values.

- a) "OK" — message successfully sent;
- b) "TXABORT" — sending process failed;
- c) "FLUSHED" — message has been removed from the pending queue before sending.

NOTE 1 Txstatus is a local variable so the coding is not specified.

NOTE 2 The FLUSHED status is only used in response to a deletion requested by the Queue maintenance service.

NOTE 3 The parameter value OK is not an indication that the message has been received.

The receiving station whose MAC ID matches that in the address, shall pass each correctly received data unit to its local DL user, together with the data unit size, the Lpacket service type and its source ID obtained from the DLPDU which conveyed the Lpacket.

6.7 Keeper UCMM Lpacket

6.7.1 General

Devices that implement the Keeper object shall also support sending and receiving of Keeper UCMM Lpackets. Only the master Keeper shall send the Keeper UCMM Lpacket.

NOTE Using a specific address for Keeper objects reduces the decoding activity required by devices which do not include a Keeper object.

6.7.2 Structure of the Keeper UCMM Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the Keeper UCMM fixed tag and 0xFF the broadcast address. The link data field shall contain the user data provided by the Keeper object.

6.7.3 Sending and receiving the Keeper UCMM Lpacket

The Keeper UCMM Lpacket shall only be sent by a station with its Keeper object in the master state. It shall be requested at the QoS priority specified by the invoking Keeper object.

The status of each Keeper UCMM transmission is confirmed to the DL user by returning the DLS-user-id (an internal handle or identifier) and the DLS Txstatus with one of the following values.

- a) "OK" — message successfully sent;
- b) "TXABORT" — sending process failed;

c) “FLUSHED” — message has been removed from the pending queue before sending.

NOTE 1 Txstatus is a local variable so the coding is not specified.

NOTE 2 The FLUSHED status is only used in response to a deletion requested by the Queue maintenance service

NOTE 3 The parameter value OK is not an indication that the message has been received

The receiving station whose MAC ID matches that in the address, shall pass each correctly received data unit to its local DL user, together with the data unit size, the Lpacket service type and its source ID obtained from the DLPDU which conveyed the Lpacket.

NOTE 4 As this is an unconnected message, the source MAC ID is required to enable a response as appropriate.

6.8 TUI Lpacket

6.8.1 General

A Table Unique Identifier (TUI) is used by all ControlNet objects to reference a consistent set of link configuration attributes. Two separate sets of attributes are held, each with their own TUI, one for current operation and one for pending operation. Changes from TUI data in the current link configuration to the TUI data in the pending link configuration shall occur at the completion of a synchronized parameter change.

The TUI Lpacket is published as a scheduled Lpacket by the master Keeper to enable all attached devices to verify they have current configuration and schedule data.

6.8.2 Structure of the TUI Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the TUI fixed tag and 0xFF the broadcast address. The link data field shall contain the listed items shown in Table 12.

Table 12 – Format of the TUI Lpacket

Name	Data type	Description of parameter	Semantics of values
Size	USINT	Size of the Lpacket in words	0x0D
Control	USINT	Link layer Lpacket control octet	0x01 (Fixed tag Lpacket)
Fixed_Tag	USINT	Fixed tag value	0x84 (TUI Lpacket)
Destination_Mac_Id	USINT	Who receives the Lpacket	0xFF (Broadcast)
Unique_Id	UDINT	CRC of the Keeper's current attributes	Least significant octet first
Status_Flag	UINT	TUI flag values	See the Keeper object TUI attribute for details
Keeper_Mac_Id	USINT	MAC ID of the Keeper broadcasting the TUI	See the Keeper object TUI attribute for details
Reserved	USINT	Reserved for data alignment	
Net_Resource_Vendor_Id	UINT	Vendor ID of object holding Net Resource for exclusive use	
Net_Resource_Serial_Number	UDINT	Serial number of object holding Net Resource for exclusive use	
Net_Resource_Class	UDINT	Class number of object holding Net Resource for exclusive use	
Net_Resource_Instance	UDINT	Instance number of object holding Net Resource for exclusive use	

When generating the transmitted TUI Lpacket, any reserved fields shall use the values as specified in the corresponding reserved fields of Keeper object attribute 0x0101. During the

configuration process, the programming tool shall set any reserved fields in attribute 0x0101, to zero.

6.8.3 Sending and receiving the TUI Lpacket

The requested QoS priority shall be scheduled.

When the Keeper object is in the MASTER_VERIFY, FAULTED_MASTER_VERIFY, MASTER, FAULTED_MASTER, NET_CHANGE_MASTER or NET_CHANGE_FAULTED_MASTER operating state, it shall attempt to transmit a Table Unique Identifier (TUI) Lpacket as scheduled data once every 4 NUTs (on NUT numbers 0, 4, 8, 12, 16 ...). All Keeper devices shall reserve 26 octets of scheduled link transmit time once every 4 NUTs for transmitting this Lpacket. (If sent unscheduled, it shall be the highest QoS priority unscheduled information.)

Reception of TUI Lpackets shall be enabled at power up by the ControlNet object in each device. Received TUI Lpackets shall be passed to the local ControlNet object.

6.9 Link parameters Lpacket and tMinus Lpacket

6.9.1 General

To synchronize the changing of local link parameters, Station Management shall enable the reception of two fixed tag Lpackets: 0x15 (tMinus) and 0x81 (link parameters) via the DLL_enable_fixed_request service of the DLL.

All nodes on a link shall maintain two copies of link parameters: current and pending. The current copy of link parameters shall be used for the on going operation of the Data Link Layer. The pending copy shall be maintained to allow a synchronized change of link parameters. The change synchronization is supported by the tMinus service.

6.9.2 Structure of link parameters and tMinus Lpackets

The size and control fields shall follow the fixed tag format. The address shall be formed from the fixed tag for the service and 0xFF the broadcast address. The link data field for the appropriate Lpacket shall contain the following contents in the listed order.

a) Contents of the Link parameters Lpacket data field:

```
class LinkParm_Lpacket : public fixedLpacket
{
public:
    UINT  NUT_length;    // the length of the NUT in 10 µs increments;
    USINT smax;          // highest MAC ID allowed to transmit scheduled
    USINT umax;          // highest MAC ID allowed to transmit unscheduled
    USINT slotTime;      // time allowed for line turnaround in 1 µs increments
    USINT blanking;      // time to disable RX after DLPDU in 1,6 µs increments
    USINT gb_start;      // 10 µs intervals from start of guardband to tone
    USINT gb_center;     // 10 µs intervals from start of moderator to tone
    UINT  zero;          // reserved
    USINT modulus;       // modulus of the interval counter
    USINT gb_prestart;   // transmit cut-off, 10 µs intervals before tone
    UDINT unique_id;     // 32 bit CRC calculated from Keeper configuration table
    UINT  status_flag;   // 16 bit status table set by the Keeper
    UINT  reserved[8];   // all zeros
};
```

NOTE The format of the instance attributes 0x80 and 0x81 of the ControlNet object are exactly the same as the link data of a fixed tag 0x81 Lpacket.

b) Contents of the tMinus Lpacket data field:

```
class tMinus_Lpacket : public fixedLpacket
{
public:
    USINT  start_count;
    USINT  reserved[3];
};
```

where start count is a number of NUTs to be counted before the transition and shall not be less than 8 and the reserved field shall contain zeros.

6.9.3 Sending and receiving the tMinus and Link parameters Lpackets

The Link parameters and tMinus Lpackets are issued by the master Keeper when a change is required to the link parameters. They shall be requested at the QoS priority specified by the invoking Keeper object.

Upon the reception of a LinkParm_Lpacket, each node shall load the pending copy of its link parameters using their local DLL_set_pending_request service. The pending parameters shall be loaded within 3 unscheduled transmit opportunities or within 1 s, whichever is greater. The pending link parameter attribute of the ControlNet object shall also be updated with the pending copy of the link parameters.

Upon reception of a tMinus_Lpacket, the node shall initiate a synchronized parameter change using the DLL-tMinus-start-countdown-request service passing the start_count parameter. The ControlNet object shall copy its pending link parameter attribute to its current link parameter attribute when the DLL_tminus_zero_indication signals the completion of the synchronized parameter change.

NOTE The DLL copies its own pending link parameters to its current copy at the completion of the tMinus countdown and no DLL_tminus_zero_indication is signaled at the completion of the countdown unless the completion is preceded by a DLL-tMinus-start-countdown-request.

The moderator Lpacket contains a field, called tMinus, that shall be used to synchronize the update of the current link parameters. The DLL-tMinus-start-countdown-request shall cause a node to participate in a tMinus countdown, and, if the node is the moderator, shall initialize the tMinus field of the moderator Lpacket. The moderator node shall decrement this field before transmitting each moderator until the field equals zero. When the tMinus field transitions from 1 to 0, the DLL in each node participating in the countdown shall locally generate a DLL_tminus_zero_indication to its ControlNet object which shall copy its pending link parameters into its current copy. If the tMinus field transitions to 0 from any value except 1, the countdown shall be aborted and no DLL_tminus_zero_indication shall be generated.

6.10 I'm-alive Lpacket

6.10.1 General

This service allows all nodes on a link to recognize that another node has just been reset or power cycled.

6.10.2 Structure of the I'm-alive Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the I'm alive fixed tag and 0xFF the broadcast address. The link data field shall contain the following in the listed order.

```
class Im_Alive_Lpacket : public fixedLpacket
{
public:
    USINT    Lpacket_variant;
    USINT    sourceID;
    USINT    reserved[6];
};
```

The Lpacket_variant and reserved fields shall be set to zero. The sourceID field shall be set to MAC ID of the node which transmits the Lpacket.

6.10.3 Sending and receiving I'm Alive

The QoS priority shall be specified by the invoking DL user.

When a node is first brought on-line before starting full operations, the network attachment monitor (NAM) in the new node shall broadcast at least 3 `Im_Alive_Lpackets`, subject to the rules for I'm Alive the state processing.

Receiving stations shall notify their management services.

NOTE Among the clients of these broadcasts are the higher layer connection managers of every other node on the link. The entities will abort any transaction records associated with the newly activated MAC ID and cancel any connections which had passed through the newly activated MAC ID. Since connection IDs (generic tags) are allocated dynamically, this ensures that connection ID's issued by the old device at the MAC ID are not improperly reused.

6.10.4 I'm alive state processing

A node shall only transmit one `Im_Alive_Lpacket` per NUT. To minimize the processing requirements in each node, a node which is broadcasting its three `Im_Alive_Lpackets` shall not count as a valid transmission any `Im_Alive_Lpacket` that is preceded in the NUT by more than seven other such Lpackets. These Lpackets shall be transmitted in the unscheduled portion of the NUT (QoS priority of either LOW or HIGH).

NOTE Because the timing of when nodes power up or reset relative to each other cannot be controlled, there is the possibility of an I'm alive Lpacket broadcast storm if a large number of nodes enter the I'm alive state at the same time. To minimize processing requirements, a transmission smearing and back off algorithm is recommended to monitor and control the intensity of any broadcast storm activity. Such an algorithm would spread the I'm alive Lpackets out over longer and longer periods of time, should a large number of nodes power up concurrently. If a small number of nodes power up together, the algorithm should allow them to exit the I'm alive state quickly since there cannot be a broadcast storm. An example of one such algorithm is shown in Figure 16.

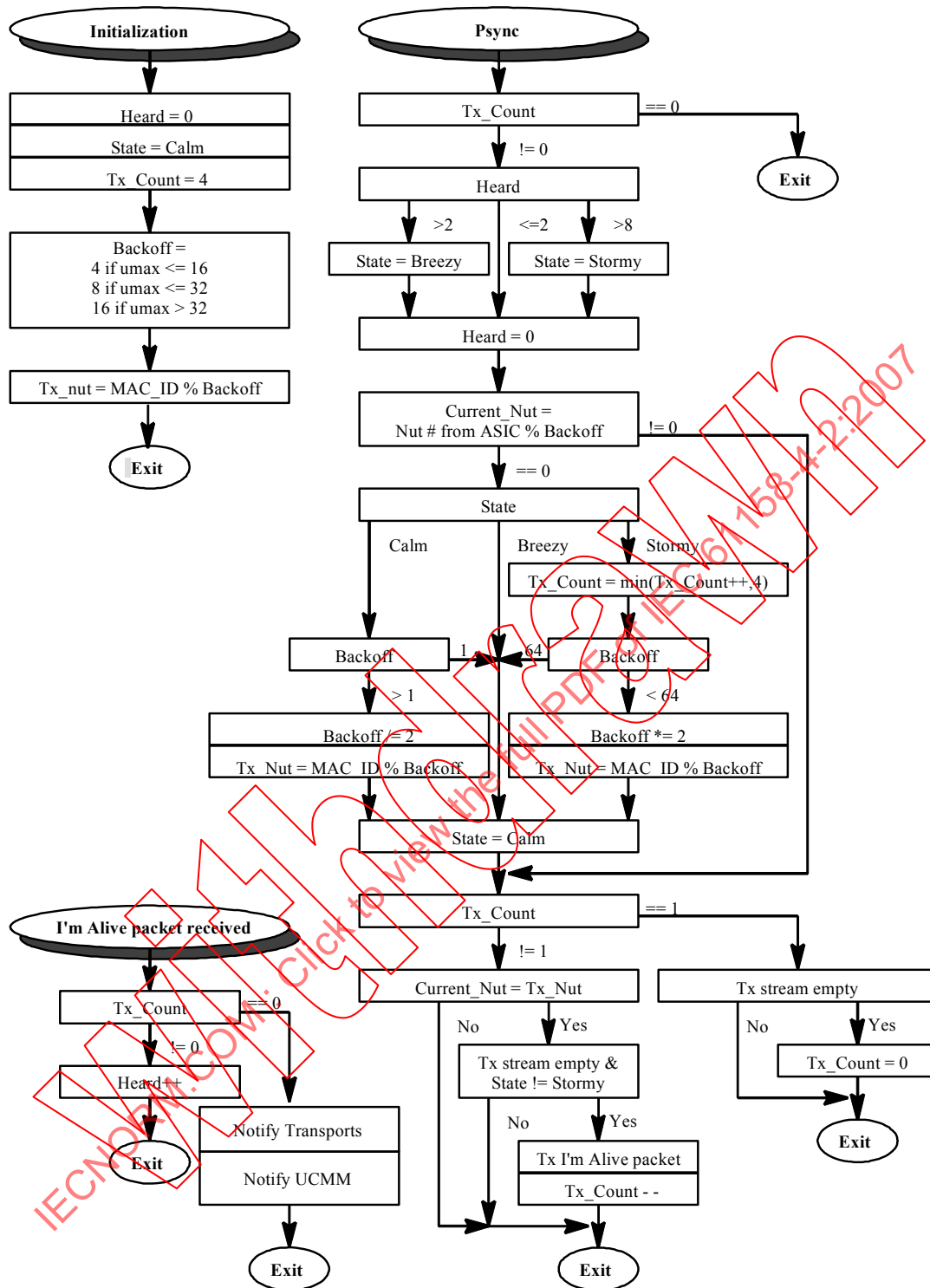


Figure 16 – Example I'm alive processing algorithm

6.11 Ping Lpackets

6.11.1 General

The ping service allows station management to initiate a ping request and receive a ping response from each active station, or one addressed station attached to the DLS provider.

All stations contain the ping service and at power-up the enable-fixed-request service is used by the DLS user to activate reception and response to ping requests within the local DLS provider.

6.11.2 Structure of the ping Lpackets

The size and control fields shall follow the fixed tag format.

- a) When sending a ping request, the address shall be formed from the ping request fixed tag and an address specified by the DL user. The specified address may be a single MAC ID or 0xFF the broadcast address.

The link data field shall contain the following in the listed order.

```
class Ping_request : public fixedLpacket
{
    public:
        USINT    client_ID;
        USINT    reserved[3];
};
```

where client_ID is the MAC ID of the request originator, and the reserved field shall contain zeros.

- b) When sending a ping reply, the address shall be formed from the ping reply fixed tag and the client_ID address from the corresponding ping request indication, (used here as destination for the reply).

The link data field shall contain the following in the listed order;

```
class Ping_reply : public fixedLpacket
{
    public:
        USINT    server_ID;
        USINT    vendor_specific[4];
        USINT    reserved[3];
};
```

where the server_ID field shall contain the MAC ID of the node responding to the request, the vendor_specific field may contain any data and the reserved field shall contain zeros.

The DLL_xmit_fixed_request(id, Lpacket, sizeof(Ping_reply), HIGH, 0x29, client_ID) service shall be used to send the reply to the MAC ID specified in the client_ID field of the received Ping_request Lpacket.

NOTE A typical use for the vendor_specific field is for debugging or diagnostic information.

6.11.3 Sending and receiving the ping Lpackets

The QoS priority for a ping request shall be specified by the invoking DL user. The QoS priority for a ping response shall be High.

Sending of ping-request Lpackets may be initiated by local management in any station.

At power up, the DLL_enable_fixed_request(id, 0x09) internal service of the Data Link Layer shall be invoked to enable the reception of fixed tag ping requests.

Upon receipt of a ping-request, each receiving station assembles a ping reply fixed tag Lpacket and sends it to the requesting station.

Upon receipt of a ping response, the requesting node passes the indication to local management.

6.12 WAMI Lpacket

6.12.1 General

A “Where Am I?” (WAMI) server shall be present in all permanent nodes which have a Network Access Port (NAP). The server shall not be present on transient nodes.

NOTE The WAMI (where am I) server allows a transient node (a node which connects to the network via the network access port or NAP) to determine the MAC ID of the node to which it is attached. This is a convenient way for software in a transient device to establish communication with the local permanent node.

6.12.2 Structure of the WAMI Lpacket

The size and control fields shall follow the fixed tag format. The address shall be formed from the WAMI fixed tag and 0xFF the broadcast address for all MAC IC nodes on this link. The link data field shall contain the following in the listed order.

```
class WAMI_Lpacket : public fixedLpacket
{
public:
    USINT    requestID;
    USINT    responseID;
    USINT    reserved[6];
};
```

In a request WAMI, the requestID shall contain the MAC ID of the transient node that is sending the Lpacket. The other fields shall contain zeros. To convert a request into a response, the server shall copy its own MAC ID into the responseID field and reply. The QoS priority is Low.

NOTE 1 After WAMI fixed tag reception is enabled by the DLL_enable_fixed_request(id, 0x86) service of the DLL, fixed tags are received by the DLL_recv_fixed_indication service.

NOTE 2 The Network Attachment Monitor (see 8.1) describes procedures to ensure the transient MAC ID does not duplicate an existing link MAC ID.

6.12.3 Sending and receiving the WAMI Lpacket

The QoS priority shall be Low.

The WAMI server is only implemented in devices having a NAP, these devices shall repeat all messages from the main link to the NAP and all messages from the NAP to the main link. Additionally these devices shall include means of detecting messages received via their NAP, and they shall only respond to Lpackets which match each of the following criteria:

- WAMI fixed tag 0x86;
- destination broadcast (0xFF);
- received via the device local NAP.

Lpackets that match these criteria shall be called request WAMI Lpackets. For all such Lpackets, the server shall generate a response WAMI that is addressed to the MAC ID which sent the request WAMI. The WAMI server shall discard all other WAMI fixed tag 0x86 Lpackets.

6.13 Debug Lpacket

This service may be used to transmit the state of an object building a trace of object state transitions on the wire.

The size and control fields shall follow the fixed tag format. The address shall be formed from the debug fixed tag and destination MAC ID provided by the DLL transmit service. The link data field shall contain the following in the listed order.

```

class Debug_Lpacket : public fixedLpacket
{
    public:
        UINT    object_class;
        UINT    data1;
        UINT    data2;
};

```

The value of `object_class` shall be the class code (see IEC 61158-6-2) of the object transmitting its state. The meaning of the `data1` and `data2` fields shall be defined in the object definition. Devices shall not generate more than one fixed tag debug Lpacket per 10 s on average so as not to interfere with other unscheduled transmissions.

6.14 IP Lpacket

This service (fixed tag 0x85) is used to send raw IP frames. The maximum IP frame size that can be sent in an Lpacket is 506 octets. Broadcast are sent with destination 0xFF. IP frames between two nodes shall have the correct fixed tag destination MAC ID. An address resolution protocol shall be used to determine the MAC ID associated with a particular IP address.

6.15 Ethernet Lpacket

This service (fixed tag 0x89) is used to send Ethernet frames other than IP frames. This includes but is not limited to Ethernet Address Resolution Protocol (ARP) frames. This allows simple implementation of IP and an Ethernet style address resolution protocol on Type 2 physical layer since an existing TCP/IP/Ethernet stack can be used with minor packet changes.

The lowest octet of the Ethernet physical address is used to represent the Type 2 MAC ID (Ethernet physical addresses are sent highest octet first). A Type 2 broadcast (MAC ID 0xFF) shall be expanded to an Ethernet broadcast address (FF FF FF FF FF FF).

On start up a TCP/IP stack that includes Ethernet ARP shall know the physical address of the local node. This is reported as "00 00 00 00 00 <MACID>".

The low octet of an Ethernet frame's "destination address" is used as the Type 2 fixed tag destination. The low octet of an Ethernet frame's "source address" will be the same as the local node's MAC ID (reported to the TCP/IP stack at startup) and is sent as the DLPDU source MAC ID. The Ethernet "frame type" (08 06 for ARP) make up the first two octets of the data portion of the Lpacket and up to 504 octets are available for the data portion of the Ethernet frame (ARP request/reply uses 28 octets).

Because the Ethernet "frame type" is included in the Lpacket, other types of Ethernet frames such as RARP can be sent using this Ethernet fixed tag service as well.

7 Objects for station management

7.1 General

This protocol specification is quite flexible. It can provide deterministic and synchronized I/O transfer at cyclic intervals up to 1 ms and node separations up to 25 km. This performance is adjustable on-line by configuring the link parameters. These parameters, which govern the access to the link, can be tuned as required to match different applications.

Station Management allows these parameters to be changed on-line, as the network is working, it also allows the link to continue functioning while connections to new nodes are added and removed.

The functions of Station Management allow

- access to variables and events within each of the Layers;

- a common user interface;
- coordinating the change of link parameters;
- non-disruptively adding nodes to the link;
- tuning of link parameters;
- clock synchronization between nodes.

The access to variables and events within each of the Layers is accomplished by defining object interfaces for these parameters.

The ControlNet object provides a consistent interface to the Physical and Data Link Layers.

The Keeper object holds the link attributes for all devices on a link and shall be responsible for distributing those attributes to those devices in an orderly fashion. It also holds (for link scheduling software) a copy of the connection originator data for all connection originator devices using a network. If there are multiple Keeper objects on a link, they perform negotiations to determine which Keeper is the master Keeper and which Keeper(s) perform backup Keeper responsibilities. The master Keeper is the Keeper actively distributing link parameters and attributes to the nodes on the network. A backup Keeper is one that monitors Keeper related network activity and can transition into the role of master Keeper should the master Keeper fail to perform.

The Keeper object also contains support for obtaining, holding, and releasing the Network Resource, a network semaphore that is used to eliminate conflicts when modifying the attribute data held by the Keeper object(s) on a link.

The Scheduling object shall exist in connection originator (CO) devices. The Scheduling object shall be used by link scheduling software (LSS) to

- read scheduled connection information;
- write schedule information;
- provide the CO with a signature (software key) which shall be used to authenticate the CO device's access to a specific link.

The TCP/IP Interface object provides the mechanism to configure the TCP/IP interface of a device.

The Ethernet Link object maintains link-specific counters and status information for a physical Ethernet ISO/IEC 8802-3 port.

The DeviceNet object provides a consistent interface to the Physical and Data Link Layers.

The Connection Configuration object provides an interface to create, configure and control connections in a device.

7.2 ControlNet object

7.2.1 Overview

The ControlNet object shall provide a consistent Station Management interface to the Physical and Data Link Layers. This object shall make diagnostic information from these layers available to client applications. Each node shall support one ControlNet object per link.

NOTE A device may contain more than one node.

7.2.2 Class attributes

The ControlNet object shall support the class attributes as specified in Table 13.

Table 13 – ControlNet object class attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x01	Required	Get	Revision	UINT	Revision of this object	First revision, value = 1
0x02	Required	Get	Max. instance	UDINT	Maximum instance number	Value determined by node specifics

7.2.3 Instance attributes

7.2.3.1 General

The ControlNet object shall support the instance attributes as specified in Table 14.

Table 14 – ControlNet object instance attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x80	Optional	Get	Pending_Link_Config	STRUCT of 34 octets	pending link configuration parameters	see 6.9.2
			Link_Config	STRUCT of 12 octets	pending link parameters	
			NUT_length	UINT	DLL NUT_length	in 10 µs ticks
			smax	USINT	DLL smax	0 to 99
			umax	USINT	DLL umax	1 to 99
			slotTime	USINT	DLL slotTime	in 1 µs ticks
			blanking	USINT	DLL blanking	in 1,6 µs ticks
			gb_start	USINT	DLL gb_start	in 10 µs ticks
			gb_center	USINT	DLL gb_center	in 10 µs ticks
			reserved	UINT	reserved	
			modulus	USINT	DLL modulus	127 required
			gb_prestart	USINT	DLL gb_prestart	in 10 µs ticks
			TUI	STRUCT of 22 octets	Keeper TUI	see 7.2.3.3
			unique_ID	UDINT	Keeper CRC	see 7.2.3.4
			status_flag	UINT	TUI flag	see 7.2.3.5
0x81	Required	Get	current_link_config	STRUCT of 34 octets	current link configuration parameters	same as attribute 0x80 see 6.9.2
0x82	Required	Get/ Get_and _Clear	diagnostic_counters	STRUCT of 42 octets	diagnostic counters	see 7.2.3.7
			buffer_errors	UINT	buffer event counter	see 7.2.3.8
			error_log	SWORD[8]	bad DLPDU log	see 7.2.3.9
			event_counters	STRUCT of 32 octets	diagnostic counters	see 7.2.3.10

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
			good_frames_transmitted	SWORD[3]	good DLPDUs transmitted (LSB first)	
			good_frames_received	SWORD[3]	good DLPDUs received (LSB first)	
			selected_channel_frame_errors	USINT	framing errors detected on active receive channel	
			channel_A_frame_errors	USINT	framing errors detected on channel A	
			channel_B_frame_errors	USINT	framing errors detected on channel B	
			aborted_frames_transmitted	USINT	DLPDUs aborted during transmission (transmit underflows)	
			highwaters	USINT	LLC transmit underflow and LLC receive overflow	
			NUT_overloads	USINT	no unscheduled time in NUT (all time used for scheduled transmissions)	
			slot_overloads	USINT	more scheduled data queued for one NUT than allowed by sched_max_frame parameter	
			blockages	USINT	single Lpacket size exceeds sched_max_frame parameter	
			non_concurrence	USINT	two or more nodes could not agree whose turn it is to transmit	
			aborted_frames_received	USINT	incomplete DLPDUs received	
			lonely_counter	USINT	number of times nothing heard on network for 8 or more NUTs	
			duplicate_node	USINT	DLPDU received from node with local node's MAC ID	
			noise_hits	USINT	noise detected that locked the modem rx PLL	
			collisions	USINT	Rx data heard just as we were going to transmit	
			mod_MAC_ID	USINT	MAC ID of the current moderator node	
			non_lowman_mods	USINT	Moderator DLPDUs heard from non lowman nodes	
			rogue_count	USINT	rogue events detected	

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
			unheard_moderator	USINT	DLPDUs being heard but no moderators being heard	
			vendor_specific	USINT	vendor specific	
			reserved	SWORD[4]	reserved	
			vendor_specific	USINT	vendor specific	
			vendor_specific	USINT	vendor specific	
			reserved	SWORD	reserved	
0x83	Required	Get	station_status	STRUCT of 6 octets	station status	see 7.2.3.11
			smac_ver	USINT	MAC implementation	
			vendor_specific	SWORD[4]	vendor specific	
			channel_state	SWORD	channel state LEDs, redundancy warning and active channel bits	see 7.2.3.12
0x84	Get required, Set-(One Time Only) Optional	Get/Set	MAC_ID	STRUCT of 4 octets	MAC ID switch and current settings	
			MAC_ID_current	USINT	current MAC ID	Range 0 to 99. See 7.2.3.13
			MAC_ID_switches	USINT	MAC ID switch settings	0 to 99, 0xFF. See 7.2.3.14
			MAC_ID_changed	BOOL	MAC ID switches changed since reset	see 7.2.3.15
			Reserved	USINT	reserved	
0x85	Optional	Get/Set	Sched_max_frame	STRUCT of 2 octets	scheduled data limit	
			Sched_max_frame	USINT	scheduled maximum DLPDU	in octet pairs. See 7.2.3.16
			Reserved	USINT	reserved	
0x86	Required	Get	error_log	STRUCT of 10 octets	driver firmware buffer error counts and troublesome node list	see 7.2.3.17
			buffer_errors	UINT	buffer event counter	
			error_log	SWORD[8]	bad DLPDU log	
0x87	Optional	Get / Get_and_Clear	extended_diagnostic_counters	STRUCT of 264 octets	additional diagnostic counters	
			unsched_transmitted	UDINT	number of unscheduled Lpackets transmitted	
			sched_highwater	UDINT	Max. number of shared/unscheduled Lpackets in transmit queue	
			sched_data	128 STRUCTS of 2 octets (256 octets)	schedule information	

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
			words_in_use	USINT	number of scheduled words in use by this node this NUT	
			Lpkt_in_use	USINT	number of scheduled Lpackets in use by this node this NUT	
0x88	Optional	Get	Active_node_table	ARRAY of 32 octets	one bit for each MAC ID 1 = node present 0 = node missing	see 7.2.3.18
0x89	Optional	Get	New_node_table	ARRAY of 32 octets	one bit for each MAC ID 1 = node has joined link recently 0 = node has not joined link recently	see 7.2.3.19
where LSB = least significant byte/octet.						

7.2.3.2 pending_link_config

The values within the attribute allow the pending link parameters to be read. This attribute shall change based on the Station Management function called synchronized parameter change, and shall not be set via the services of the ControlNet object.

7.2.3.3 Keeper TUI

The ControlNet object shall maintain separate copies of the TUI sub-attribute for pending_link_config and current_link_config attributes. Changes to the TUI data in the pending_link_config attribute shall occur at the reception of a fixed tag 0x81. Changes to the TUI data in the current_link_config attribute shall occur at the completion of the synchronized parameter change.

7.2.3.4 unique_ID

The unique_ID field shall consist of a 32 bit CRC value calculated from the contents of the network configuration table maintained by the Keeper. This value shall be copied from the fixed tag 0x81 Lpacket.

7.2.3.5 status_flag

The status_flag field of the TUI sub-attribute shall consist of a 16 bit value. Unused bits shall be reserved and set to zero.

As shown in Table 15, the Keeper State bit (bit 4) shall indicate whether this node has ever heard a TUI from a master Keeper. The ControlNet object shall clear this bit at initialization. Since all TUI Lpackets sent by the master Keeper have this bit set, the ControlNet object need not manipulate this bit.

Reception of TUI Lpackets shall be enabled at power up using the DLL_enable_fixed_request(id, 0x84) service. The status field of the TUI Lpackets shall be checked. If bit3 = 0, bit4 = 1, and bit5 = 1, then the TUI Lpacket shall be copied into attribute 0x81 and TUI reception shall be disabled using the DLL_disable_fixed_request(id, 0x84). Bits 0 and 1 of the TUI status field shall be used to override the default network redundancy settings for the node. They shall indicate if the network is being operated single channel or redundant channel mode. Only network redundancy information from TUI Lpackets with both the Keeper State and Keeper configured bits set, shall be used.

Table 15 – TUI status flag bits

Bit	Meaning
0 - 1	Network Redundancy Bit 1 = 0, bit 0 = 0; Illegal combination Bit 1 = 0, bit 0 = 1; Channel B only Bit 1 = 1, bit 0 = 0; Channel A only Bit 1 = 1, bit 0 = 1; Redundant
2	Holding Network Resource bit 0 = Network Resource not being held 1 = Network Resource being held for the object identified in the TUI
3	Network change in progress bit 0 = No network change in progress 1 = Network change in progress
4	Keeper State bit 0 = TUI transmitted by Keeper not in MASTER state 1 = TUI transmitted by Keeper in MASTER state
5	Keeper configured bit 0 = No master Keeper heard since joining link 1 = Heard master Keeper since joining link
6 - 15	reserved (set to 0)

7.2.3.6 current_link_config

The values within the `current_link_config` attribute correspond to the link parameters currently used. The format of this attribute shall be identical to that of the `pending_link_config` attribute.

7.2.3.7 diagnostic_counters

The `diagnostic_counters` attribute shall maintain counts of events in the Physical and Data Link Layers.

NOTE The `diagnostic_counters` attribute may be read without effecting their values by using one of the `Get_Attribute` service requests. The information may be read and the values set to zero by using the `Get_and_Clear` service request. See 7.2.5.

The event information maintained by the firmware shall also be gettable (but not clearable) via the `error_log` attribute. This is done for efficiency. The information maintained by the firmware shall be immediately and directly accessible.

7.2.3.8 buffer_errors

The value of the `buffer_errors` sub-attribute shall increment on every receive overflow event and transmit underflow event. If an implementation never overflows or underflows, `buffer_errors` shall be zero.

7.2.3.9 error_log

If `DLL_event_indication` returns a value of `DLL_EV_badFrame`, the `error_log` sub-attribute records the last eight such indications. The `mac_id` parameter of this indication shall be recorded. The order of MAC IDs shall be most recent error first to oldest error last. Values of zero represent unused positions in the error log.

7.2.3.10 event_counters

The `event_counters` sub-attribute shall contain counts of link events. The counter shall rollover when their values exceed their limits.

7.2.3.11 station_status

This attribute shall be only gettable.

7.2.3.12 channel_state

The channel_state octet shall represent the current state of the network status indicators, if implemented. The bits of the channel_state octet are described in Table 16.

NOTE The network status indicators are described in Annex A.

Table 16 – Channel state bits

Bits	Description
0,1,2	Channel A LED State 0 = Off 1 = Solid green 2 = Flashing green-off 3 = Flashing red-off 4 = Flashing red-green 5 = Railroad red-off 6 = Railroad red-green 7 = Solid red
3,4,5	Channel B LED State Indications same as channel A LED.
6	Redundancy Warning – When warning, the non-active channel in a redundant configuration is unusable by the controller. 0 = normal (no warning) 1 = warning
7	Active Channel – Indicates which of two channels the receiver is currently listening to. 0 = Channel B 1 = Channel A

7.2.3.13 MAC_ID_current

The MAC_ID_current value shall be accessible using the Get_Attribute services. Setting shall be required on devices without address switches. Only the first set, of the MAC_ID_current value, after a ControlNet object reset shall be accepted. Setting shall be optional on devices with address switches.

The range of allowable values for MAC IDs shall be 1 to 99.

7.2.3.14 MAC_ID_switches

The MAC_ID_switches value shall be gettable and not settable on all devices. The returned value shall be 0xFF for auto-address nodes and nodes without address switches.

7.2.3.15 MAC_ID_changed

The MAC_ID_changed value shall be cleared when the device is reset and set whenever the device detects that the address switches have been changed. Once set, it stays set until the device is reset, even if the address switches are returned to their original settings. The frequency at which the address switches are checked shall be device dependent.

When a change in the network address switches is detected, this flag shall be set and a minor error shall be reported to the host.

7.2.3.16 sched_max_frame

The sched_max_frame attribute shall be used to limit the maximum size of the node's scheduled transmissions. A value of 0 shall indicate that a node shall not transmit during the scheduled portion of the NUT.

7.2.3.17 error_log

The instance attribute 0x86 shall be a copy of the error_log sub-attribute contained in instance attribute 0x82.

NOTE The information in this attribute is a copy of the information found in the diagnostic_counters attribute. This copy is only gettable. The values in this attribute may be cleared only by using the Get_and_Clear service request on the diagnostic_counters attribute. See 7.2.5 for more details.

This copy of the error_log shall only be accessible via the Get Attribute services. This attribute shall only be cleared when the master copy in 0x82 is cleared.

7.2.3.18 Active node table (instance attribute = 0x88)

Instance attribute 0x88 of the ControlNet object shall consist of an array of 256 bits, one per MAC ID. The least significant bit shall correspond to MAC ID = 0; the most significant bit shall correspond to MAC ID = 255. The bit for a specific MAC ID shall be set (1) for a node within 1 s of a node transmitting on the link. The bit shall be cleared (0) within 20 s of the node leaving the link. A node shall be considered to have left the link if it misses three consecutive transmit opportunities.

7.2.3.19 New node table (instance attribute = 0x89)

Instance attribute 0x89 of the ControlNet object shall consist of an array of 256 bits, one per MAC ID. The least significant bit shall correspond to MAC ID = 0; the most significant bit shall correspond to MAC ID = 255. The bit for a specific MAC ID shall be set (1) for a node within 1 s of a node transmitting on the link. The bit shall be cleared (0) between 10 s and 20 s after the node has joined the link.

7.2.4 Common services

7.2.4.1 General

The ControlNet object shall support the common services as specified in Table 17.

Table 17 – ControlNet object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x05	N/A	Required	Reset	Return ControlNet object to power up state
0x10	N/A	Conditional	Set_Attribute_Single	Set network or node specific configuration information
0x0E	Required	Required	Get_Attribute_Single	Get network or node specific configuration information
0x01	N/A	Optional	Get_Attribute_All	Get network and node specific configuration information
0x02	N/A	Optional	Set_Attribute_All	Set network and node specific configuration information
0x03	N/A	Optional	Get_Attribute_List	Get network and/or node specific configuration information
0x04	N/A	Optional	Set_Attribute_List	Set network and/or node specific configuration information

7.2.4.2 Reset service

The Reset service shall reinitialize the Network Attachment Monitor (NAM) causing it to monitor the link activity and then attempt to join the link. MAC ID switches shall not be re-evaluated due to the reset.

The ControlNet object need not respond to a Reset request before actually performing the reset operation.

7.2.4.3 Set_Attribute_Single

If any attribute has been implemented as settable, the Set_Attribute_Single service shall be required.

7.2.4.4 Get_Attribute_All response

The object/service specific reply data portion for a successful Get_Attribute response is shown in the Instance Attributes for the ControlNet object table. The size of this response is dependent on the attribute instance(s) being accessed.

Because of the large size of the Get_Attribute_All response, devices with limited resources need not support this service.

The data portion of a Get_Attribute_All service request shall contain the following attributes in the following order:

- 0x81 - current_net_config
- 0x82 - diagnostic_counters
- 0x83 - station_status
- 0x84 - MAC_ID
- 0x86 - error_log

7.2.5 Class specific services

7.2.5.1 General

The ControlNet object shall support the class specific services as specified in Table 18.

Table 18 – ControlNet object class specific services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x4C	N/A	Required	Get_and_Clear	Get then clear the diagnostic counters
0x4D	N/A	Required	Enter_Listen_Only	Force the node to enter and stay in listen only mode until reset
0x4E	N/A	Required (See 7.2.5.4)	Where_am_I	Determine the MAC ID of the device this node is attached via the NAP
0x4F	N/A	Conditional	Auto_Address	Required in all auto-address nodes. Select a new MAC ID using the auto address sequence of the NAM

7.2.5.2 Get_and_Clear service

This class specific service shall be evoke an identical response to the Get_Attribute_Single service; however, after the response Lpacket is built, the value of the attribute shall be set to zero. Only instance attributes 0x82 and 0x87 shall support this service.

7.2.5.3 Enter_Listen_Only service

The Enter_Listen_Only service shall cause this object to send a `DLL_listen_only_request(FALSE)` to the DLL associated with this instance of the ControlNet object. This service need not generate a response since the node may stop transmitting before the response is sent.

NOTE 1 The node should remain in the listen-only state until a reset request is sent to either the ControlNet or Identity object.

NOTE 2 The Enter_Listen_Only service is intended to be used for diagnostic purposes to force nodes to drop off the link until they are specifically requested to rejoin. This allows a node to effectively be removed from the link without having to physically disconnect the network cable.

7.2.5.4 Where_am_I service

All transient nodes shall implement the Where_am_I service. This service shall determine the MAC ID of the node to which the transient node is connected via its network access port (NAP). The service may be implemented by using the WAMI server functionality of the Station Management entity.

NOTE A transient node may use this service to identify the MAC ID of the node into whose NAP it is plugged.

The Where_am_I response shall be a single USINT containing the MAC ID of the permanent node to which the transient node is connected. If the ControlNet object is unable to complete the WAMI query, it shall return a MAC ID of 255.

7.2.5.5 Auto_Address service

For nodes that support automatically selecting a MAC ID, this service shall reinitialize the NAM causing it to monitor the link activity and then attempt to join the link possibly with a new MAC ID. Nodes that do not support automatically selecting a MAC ID shall not implement this service.

The Auto_Address service need not respond prior searching for a new MAC_ID.

7.2.6 Behavior

7.2.6.1 Effect of DLL_tminus_zero_indication

Upon receipt of this indication, the object shall copy the contents of instance attribute 0x80 (pending_link_config) into instance attribute 0x81 (current_link_config). An internal copy of

instance attribute 0x80 shall be kept even if the implementation has not made it accessible via one of the Get_Attribute services.

7.2.6.2 Duplicate node

If a `DLL_event_indication(DLL_EV_dupNode)` is received from the Data Link Layer, the ControlNet object shall force the DLL into a listen-only state. The ControlNet object shall latch this state until a reset service is received. A reset to the Identity object shall also end the listen only state.

7.2.6.3 Redundancy

The network redundancy setting for a node shall be initially set at start up to the best capability of the device. (Channel A only for single channel devices, both channels for network redundant devices). The power up redundancy setting shall be reflected in the TUI contained in the instance attribute 0x82.

The network redundancy setting for a node shall be updated to the current network redundancy setting by the Keeper in one of two ways:

- The node receives a TUI Lpacket from the wire with the master Keeper and configured Keeper bits set and the net change in progress bit reset in the Status_Flags word;
- The node receives a fixed tag 0x81 Lpacket.

In either case, the received TUI data shall be copied into attribute 0x80.

If a single channel node joins a redundant link, it shall behave as if it were a redundant node with a faulty channel.

7.2.7 Module status indicator

The module status indicator, if implemented, shall provide the following status indications:

- a) solid green when connections are active in the *on line* state;
- b) flashing green/off in the *check for moderator listen only* state;
- c) flashing green/off when rogue conditions are detected;
- d) flashing red/off for a recoverable error;
- e) flashing red/off when in the `DUP_LISTEN_ONLY` state;
- f) solid red for an irrecoverable error.

7.3 Keeper object

7.3.1 Overview

The Keeper object shall hold the link attributes for all devices on a link and shall be responsible for distributing those attributes to those devices in an orderly fashion. It also holds (for link scheduling software) a copy of the connection originator data for all connection originator devices using a network. If there are multiple Keeper objects on a link, they perform negotiations to determine which Keeper is the master Keeper and which Keeper(s) perform backup Keeper responsibilities. The master Keeper is the Keeper actively distributing attributes to the nodes on the network. A backup Keeper is one that monitors Keeper related network activity and can transition into the role of master Keeper should the master Keeper fail to perform.

The Keeper object also contains support for obtaining, holding, and releasing the Network Resource, a network semaphore that is used to eliminate conflicts when modifying the attribute data held by the Keeper object(s) on a link.

7.3.2 Revision history

The revision history of the Keeper object is described in Table 19.

Table 19 – Keeper object revision history

Class revision	Description of changes
01	Initial Release
02	Release for IEC 61158 series

7.3.3 Class attributes

The Keeper object shall support the class attributes as specified in Table 20.

Table 20 – Keeper object class attributes

Number	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x1	Required	Get	Revision	UINT	Revision of this object	2 (revision 2) (See Note 2)
NOTE 1 Revision 1 Keeper devices are restricted to only operate at MAC ID = 1. These devices perform the functions of the Keeper as described in this specification as long as no other Keeper is present on the Network. At any other MAC ID, these devices containing revision 1 Keepers perform as if no Keeper objects were present.						
NOTE 2 Revision 2 Keepers devices operate at any legal MAC ID value. These devices perform the functions of the Keeper as described in this specification in the presence of any other Revision 2 Keepers on the Network.						

7.3.4 Instance attributes

7.3.4.1 General

The Keeper object shall support the instance attributes as specified in Table 21. Only one instance of the Keeper object is allowed. That instance shall be sensitive to the port on which requests are received. This is especially important in the case of a router device where one Keeper instance receives requests from two links.

Table 21 – Keeper object instance attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x00	Required	Get	status	STRUCT of 2 octets	Keeper object status	
			state	USINT	current Keeper operating state	
			reserved	USINT	reserved for data alignment	
0x01 through 0x63	Required	Get/Set	port_status	STRUCT of 10 octets	port status for node 1 through node 99	
			port_status	UINT	port status	
			ID	STRUCT of 8 octets	node type identification	
			vendor	UINT	vendor code	
			type	UINT	product type	
			code	UINT	product code	
			major	USINT	major revision	
			minor	USINT	minor revision	

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x64 – 0xFE	Reserved					Reserved for future expansion
0xFF	Required	Get/Set	net_config	STRUCT of 12 octets	current network parameters	
			NUT	UINT	Network Update Time	In 10 μ s ticks
			smax	USINT	Scheduled max node ID	0 to 99
			umax	USINT	Unscheduled max node ID	0 to 99
			Slot_Time	USINT	Slot Time	In 1 μ s ticks
			Blank_Time	USINT	Blanking Time	In octets
			Gb_Start	USINT	Guard Band Start	In 10 μ s ticks
			Gb_Center	USINT	Guard Band Center	In 10 μ s ticks
			Reserved	UINT	Reserved for Data Alignment	
			Int_Cnt mod	USINT	Interval Count Modulus (Macrocycle length)	127
0x100	Required	Get/Set	Gb_Prestart	USINT	Guard Band Prestart	In 10 μ s ticks
			name	UINT [33]	current name of this link	32 unicode characters and a unicode NULL
0x101	Required	Get/Set	RT_TUI	STRUCT of 22 octets	Table Unique Identifier	
			unique_ID	UDINT	attribute CRC	
			status_flag	UINT	TUI flag bits	See 7.2.3.3
			reserved	USINT [16]	reserved to allow common format with attribute 0x102	
0x102	Required	Get	Link_TUI	STRUCT of 22 octets	Table Unique Identifier (current link only)	
			unique_ID	UDINT	attribute CRC	
			status_flag	UINT	TUI flag bits	See 7.2.3.3
			Keeper_MAC_ID	USINT	MAC ID of node broadcasting the TUI	Any legal MAC ID
			reserved	USINT	reserved for data alignment	
			Net_Resource_Vendor_Id	UINT	Vendor ID of object holding Net Resource	
			Net_Resource_Serial_Number	UDINT	Serial number of object holding Net Resource	
			Net_Resource_Class	UDINT	Class of object holding Net Resource	
			Net_Resource_Instance	UDINT	Instance of object holding Net Resource	
0x103	Required	Get/Set	Cable_Config	STRUCT of up to 100 octets	Current cable configuration	
			Id	USINT	Id value	Always 0xFF
			Num_Elements	USINT	Number of cable configuration elements in network configuration	Range 1 to 24

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
			Propagation_time	UINT	Number of 100 ns ticks	
			Physical element	Phy element [24]		
			Phy_element	STRUCT of 4 octets		
			Vendor_id	UINT	Vendor code	
			Product_code	USINT	Product code	
			How_many	USINT		
0x104	Required	Get/Set	CO_summary	STRUCT of 204 octets	General information about the <i>co_data</i> attribute	
			data_size	UINT	Size of <i>co_data</i> attribute	In words
			connection_info_revision	USINT		Used by programming tool 0 = Format 1 1 = Format 2 2 - 255 reserved
			tool_keeper_revision	USINT	Highest level <i>co_data</i> parse rule supported	Level supported is ≤ Keeper object revision 0 = Format 1 1 - 255 reserved See 7.3.6.7
			Offsets	UINT[100]	Array of word offsets into the <i>co_data</i> attribute, by MAC ID	In words 0xFFFF = no data for this MAC ID node 0 based index
0x105	Required	Get/Set fragment	CO_data	STRUCT of up to 14 988 octets	connection originator information	Tool Keeper Revision Format 1
			Branch	STRUCT of variable size	Information relative to all routers or connection originators on this link	
			num_devices	UINT	number of devices on this link	
			device	STRUCT of variable size	details of this device	
			type	USINT	types of device	1 = router 2 = connection originator
			Path_Size	USINT	size of path to node	In 16 bit words
			Path	ARRAY of UINT	path to device	
			CO_data	STRUCT of variable size	Details of CO data	Present only for CO devices (type = 2)

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
			COP	UDINT	CO password, as provided to the CO at the end of a scheduling session (see 7.4.5.8)	
			Size	UINT	Size of Connection data	In 16 bit words
			Connection	STRUCT of variable size	Connection data	Connection Info Revision in the format specified by connection_info_revision of attribute 0x104 One per connection for this CO device. See 7.3.4.7

7.3.4.2 Operating state definitions

The values for the Keeper operating state shall be as defined in Table 22.

Table 22 – Keeper operating state definitions

Value	Description
0	Power up - Not on line
1	Power up - TUI wait
2	Power up - TUI poll
3	Backup
4	Master verify
5	Master
6	Faulted Backup
7	Faulted Master verify
8	Faulted Master
9	Net Change - Backup
10	Net Change - Master
11	Net Change - Faulted Backup
12	Net Change - Faulted Master

7.3.4.3 Port status flag bit definitions

The values for the Port status flag shall be as defined in Table 23.

Table 23 – Port status flag bit definitions

Bit	Meaning
0	Node accepts scheduled connections bit 0 = Node does not accept scheduled connections 1 = Node accepts scheduled connections
1 – 15	reserved (set to 0)

7.3.4.4 status_flag

The Keeper configured bit cleared in the TUI status word shall identify a Keeper with an invalid configuration or cleared memory.

Referring to Table 24, the status_flag field of the TUI sub-attribute shall consist of a 16 bit value. Unused flag bits shall be reserved and set to 0.

The holding network resource bit shall only be used by the Keeper object.

The Master Keeper bit shall indicate whether or not this node has ever heard from a master Keeper. The ControlNet object shall clear this bit at initialization. All TUIs sent by the master Keeper shall have this bit set. If a TUI Lpacket is received with the Master Keeper bit not set, the Lpacket shall be retained as a received TUI, but the redundancy information shall not be used.

The redundancy bits shall be used to override the default redundancy settings for the node. They shall indicate if the link is being operated in the single channel mode or redundant. The redundancy information shall not be used if a TUI Lpacket is received with the Master Keeper bit clear.

The net change in progress bit shall indicate that a synchronized network change sequence is in progress and that new nodes shall delay going on-line until the change sequence is completed and the bit is cleared. This bit shall be cleared in the case of a network resource timeout situation or if the network resource is released.

Table 24 – TUI status flag bits

Bit	Meaning	Used in:		
		RT_TUI	Network_TUI	ControlNet object TUI
0 - 1	Network Redundancy Bit 1 = 0, bit 0 = 0; Illegal combination Bit 1 = 0, bit 0 = 1; Channel B only Bit 1 = 1, bit 0 = 0; Channel A only Bit 1 = 1, bit 0 = 1; Redundant	X	X	X
2	Holding Network Resource bit 0 = Network Resource not being held 1 = Network Resource being held for the object identified in the TUI		X	
3	Network change in progress bit 0 = No network change in progress 1 = Network change in progress	X	X	X
4	Keeper State bit 0 = TUI transmitted by Keeper not in MASTER state 1 = TUI transmitted by Keeper in MASTER state.		X	X
5	Keeper configured bit 0 = Keeper attributes not configured (RT_TUI) 1 = Keeper attributes configured (RT_TUI)	X	X	X
6 - 15	reserved (set to 0)			

7.3.4.5 unique_ID

The unique_ID field of the TUI attribute shall consist of a 32-bit value calculated from the contents of the Keeper attribute table (excluding the TUI attribute) by the software configuration tool. The following attributes shall be used, in the given order to calculate the TUI unique_ID:

0x1 through 0x63 (port parameters for nodes 1 through 99)

0xFF (link parameters)

0x100 (link name);

0x101 (RT TUI) redundancy bits only;

0x103 (cable configuration);

0x105 (CO device password data only);

The TUI unique identifier shall be calculated as follows:

- the TUI unique identifier shall be initialized to 0;
- the data for attributes 0x1 through 0x63, attribute 0xFF, attribute 0x100, redundancy setting (attribute 0x101 RT_TUI status flag bits 0–1), 0x103 and each CO password (in address order) shall be run through the CRC polynomial.

The polynomial used to calculate the CRC shall be

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + 1$$

as shown in Figure 17.

```
UDINT CRC( // (r) the updated CRC
    unsigned long CRC, // (i) the CRC to start with
    SWORD* buffer, // (i) pointer to buffer of SWORD / octets
    WORD size) // (i) the number of octets in the buffer

# define KEEPER_CRC_POLY 0xEDB88320L
WORD octetIndex;
WORD bitIndex;
SWORD oneOctet;

// loop through the octets passed, including them in the CRC
for (octetIndex=0; octetIndex < size; octetIndex++)
{
    oneOctet = buffer[octetIndex];

    // loop through the bits passed, including them in the CRC
    for (bitIndex = 0; bitIndex < 8; bitIndex++)
    {
        if ((oneOctet & 0x1) ^ (CRC & 0x1))
        {
            CRC = (CRC >> 1) ^ KEEPER_CRC_POLY;
        } else {
            CRC = (CRC >> 1);
        }
        oneOctet >>= 1;
    }
}

// return the new CRC
return(CRC);
}
```

Figure 17 – Keeper CRC algorithm

7.3.4.6 Keeper MAC_ID

The Keeper_MAC_ID should be the MAC ID of the current master Keeper that is broadcasting the TUI. If no TUI has been received in the last 12 NUTS, a value of 0xFF shall be returned.

7.3.4.7 Connection

Various formats exist for connection data. Each format is described below. The entire collection of connection data entries shall be an integral number of words.

Format 1 does not specify the size of the path; therefore, its path[] shall be one of two forms:

- class segment, instance segment, then two connection point segments; or
- an ANSI extended symbol segment.

```
// Format 1
USINT  O2T_Size      // in 16 bit words
UINT   O2T_Rpi       // in 1 ms ticks from 2 to 32 767
USINT  T2O_Size      // in 16 bit words
UINT   T2O_RPI       // bit 15 = connection point, 0 = point to point, 1 = multipoint
// bit 14 = API in ms from 2 to 32 767
USINT  O2T_Nui_Api   // The highest bit set indicates the API;
// bit 7 = 128*NUT, bit 6 = 64*NUT, etc.
// Remaining bits indicate the O2T_NUI. If API = 1, NUI = 0
USINT  T_Node        // Range 1 - 99
USINT  T2O_Nui_Api   // The highest bit set indicates the API;
// bit 7 = 128*NUT, bit 6 = 64*NUT, etc.
// Remaining bits indicate the O2T_NUI. If API = 1, NUI = 0
UINT   path[]        // forward open connection path from this
```

```
// Format 2
UINT   O2T_net_params // copied from forward open
UINT   T2O_net_params // copied from forward open
UINT   O2T_RPI        // RPI in floating point format
UINT   T2O_RPI        // RPI in floating point format
USINT  O2T_schedule   // from schedule segment in forward open
USINT  MAC_ID         // from schedule segment in forward open
USINT  T2O_schedule   // from schedule segment in forward open
USINT  path_size      // in 16-bit words
UINT   path[]         // forward open connection path from this
```

```
UDINT convert_to_10us_ticks (UINT RPI)
{
    return (RPI <= 0x4000) ? RPI :
           (RPI <= 0xE000) ? (RPI & 0x1FFF) + 0x2000 << (RPI>>13) - 1 :
                           : (RPI & 0x0FFF) + 0x1000 << (RPI>>12 & 1) + 7;
}

UINT convert_to_Keeper_format (UDINT ticks)
{
    if (ticks < 0x00004000) return RPI;
    if (ticks < 0x00008000) return RPI>>1 & 0x1FFF;
    if (ticks < 0x00010000) return RPI>>2 & 0x1FFF;
    if (ticks < 0x00020000) return RPI>>3 & 0x1FFF;
    if (ticks < 0x00040000) return RPI>>4 & 0x1FFF;
    if (ticks < 0x00080000) return RPI>>5 & 0x1FFF;
    if (ticks < 0x00100000) return RPI>>7 & 0x0FFF;
    if (ticks < 0x00200000) return RPI>>8 & 0x0FFF;
    return 0xFFFF;
}
```

7.3.4.8 Attribute data

A Keeper maintains the link and node configuration information for its link in an attribute data structure. The Keeper shall keep two copies of the attribute structure: pending copy in RAM and a current copy in non-volatile memory. When new attributes are set in the Keeper, they shall be set in the pending RAM copy. The RAM copy shall be written to the currently active copy in non-volatile storage only upon receipt of a special request to do so.

The attributes shall use the following rules for getting and setting:

- Set service requests set the pending copy in RAM;
- Get service requests get the current copy in non-volatile storage.

Keeper shall maintain the collection of attributes specified in Table 25:

Table 25 – Keeper attributes

Attribute number	Keeper attributes
0x01 - 0x63	(port parameters for nodes 1 through 99)
0xFF	(link parameters)
0x100	(link name)
0x101	(Keeper TUI)
0x103	(cable configuration)
0x104	(node offset information into CO data)
0x105	(CO path and password information)

All Keeper attribute definitions assume a maximum of 99 nodes on the link. The Keeper object shall have memory to store the CO_summary and CO_data attributes.

Memory requirements (in octets) for the Keeper attributes are as specified in Table 26.

Table 26 – Memory requirements (in octets) for the Keeper attributes

Octets	Keeper attributes
2	Keeper status
990	Port parameters (99 nodes at 10 octets each)
12	link parameters
66	link name
22	TUI
100	Cable configuration
204	CO summary
14 988	CO data (7 494 words)
16 384	Total (Exactly 16K octets)

The sequence of events required to properly and safely update the Keeper attributes shall be as follows:

- obtain the Network Resource;
- issue a Change_Start service request to the Keeper;
- issue the appropriate Set_Attribute service request(s) to update the pending copy of the Keeper attributes;
- issue a Change_Complete service request to the Keeper. Doing this shall initiate a synchronized parameter change;
- release the Network Resource.

7.3.5 Common services

The Keeper common services shall all operate with an internal time out to prevent a hardware error condition from indefinitely stalling operations. The actual time out values shall be device dependent.

The Keeper object shall support the common services as specified in Table 27.

Table 27 – Keeper object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x03	N/A	Required	Get_Attribute_List	Get network and node specific configuration information
0x04	N/A	Required	Set_Attribute_List	Set network and node specific configuration information
0x0E	Required	Required	Get_Attribute_Single	Get network and node specific configuration information
0x10	N/A	Required	Set_Attribute_Single	Set network and node specific configuration information

The Keeper object is capable of receiving these service requests via fixed tags 83 and 88. When received via fixed tag 83, the request is processed and responded to as required independent of Keeper state. When received on fixed tag 88, the Keeper will only respond to the request if it is in the Master, Net Change Master, Faulted Master or Net Change Faulted Master states. Keeper nodes in other states shall not respond to the request.

7.3.6 Class specific services

7.3.6.1 General

Any device that has implemented the Keeper object shall also implement the Keeper UCMM.

The Keeper object shall support the following class specific as specified in Table 28.

Table 28 – Keeper object class specific services

Service code	Need in implementation		Service name	Description of service	Parameter(s)	
	Class	Instance				
0x4B	N/A	Required	Obtain_Network_Resource (ONR)	Attempt to obtain the Network Resource for this link. If successful, hold for 60 s	UINT vendor ID UDINT serial number UDINT class UDINT instance	
0x4C	N/A	Required	Hold_Network_Resource (HNR)	Continue to hold the previously obtained Network Resource for this link for 60 s	UINT vendor ID UDINT serial number UDINT class UDINT instance	
0x4D	N/A	Required	Release_Network_Resource (RNR)	Release the Network Resource for this link	UINT vendor ID UDINT serial number UDINT class UDINT instance	
0x4E	N/A	Required	Change_Start (CS)	Copy current attributes in non-volatile storage to pending attributes in RAM. Enter the appropriate <i>net change</i> operating state	<none>	
0x4F	N/A	Required	Change_Complete (CC)	Copy pending attribute data in RAM to current data in non-volatile storage. Restart normal Keeper operation. Initiate synchronized network change if net attributes changed. If number of sets does not match, do <i>change_abort</i> processing	16 bit number of "sets" performed since <i>Change_Start</i>	
0x50	N/A	Required	Change_Abort (CA)	Discard changes made to the pending attribute data, release the network resource and return to normal Keeper operation	<none>	
0x51	N/A	Required	Get_Signature (GS)	Get signature value for a specific connection originator (CO password)	Request: path size (octet) (size is in words) path Response: 32 bit cop	
0x52	N/A	Required	Get_Attribute_Fragment (GAF)	Get a portion of an attribute specifically designed to be fragmented, that is too large to fit in a single Lpacket. For use only on the <i>co_data</i> attribute	UINT size; //words UINT offset; //offset	
0x53	N/A	Required	Set_Attribute_Fragment (SAF)	Set a portion of an attribute specifically designed to be fragmented, that is too large to fit in a single Lpacket. For use only on the <i>CO_data</i> attribute	UINT size; //words UINT offset; //offset UINT data []	

The error codes for these services shall be as specified in Table 29, where the service codes are the abbreviations for the service names specified in Table 28.

Table 29 – Service error codes

Error code	Meaning	Error code use by service								
		ONR	HNR	RNR	CS	CC	CA	GS	GAF	SAF
0x02	resource_unavailable_err	X	X	X	X					
0x03	invalid_parameter_err					X		X	X	X
0x04	path_segment_err							X	X	X
0x05	path_dest_unknown_err	X	X	X	X	X	X	X	X	X
0x08	unimplemented_service_err	X	X	X	X	X	X	X	X	X
0x09	invalid_attr_err								X	X
0x0C	wrong_object_state_err					X	X			X
0x0D	already_exists_err	X			X					
0x0F	no_permission_err		X	X						
0x11	reply_too_large_err							X		
0x13	not_enough_data_err	X	X	X		X		X	X	X
0x14	undefined_attr_err		X							
0x15	too_much_data_err							X		X
0x16	nonexistant_object_err	X	X	X	X	X	X	X	X	X
0x26	invalid_path_size_err	X	X	X	X	X	X	X	X	X

7.3.6.2 Network resource

The Network Resource (NR) shall be a link semaphore that is used to eliminate conflicts when modifying attribute information in Keeper objects.

In operation, the NR shall appear as a status flag bit and other data fields in the TUI Lpacket that shall be repetitively broadcast by the master Keeper.

A device that wishes to update the Keeper attribute information first shall submit a request to the master Keeper to obtain the NR on its behalf (via an *Obtain_Network_Resource* service). This request shall be broadcast since the identity of the master Keeper is not generally known. Once obtained, the device shall periodically ask (every 15 s) the master Keeper to continue holding the NR via a *Hold_Network_Resource* service request. The master Keeper shall maintain a 60 s timer that is started when the NR is initially obtained, and retriggered when a subsequent *Hold_Network_Resource* request from the same node is received. If the timer ever times out, the master Keeper shall automatically release the NR.

A backup Keeper shall respond to the Network Resource service requests as follows. When an *Obtain_Network_Resource* service request is received, it shall start a 60 s timer that shall be retriggered when a *Hold_Network_Resource* service request is received. The timer shall also be started when a Keeper powers up on an existing network and senses that the network resource is currently being held. The timer shall be stopped when a *Release_Network_Resource* service is received or the timer expires.

When a backup Keeper becomes the master Keeper, it shall use the data from the most recently received TUI Lpacket to hand off the holding of the NR from the old master Keeper.

In both master and backup Keepers, the 60 s timer shall be used to determine if the node requesting the NR has failed or has been prematurely removed from the network. When a Keeper's 60 s NR timer expires, the Keeper shall

- a) abort any attribute changes in progress;

- b) reset the net change in progress TUI status bit;
- c) exit the net change operating state;
- d) return to normal operation.

Since the NR can be held over a fairly long period if the user is performing substantial edits, the NR shall be cleanly handed off to the new master Keeper if the master Keeper changes. This handoff shall be invisible to the node that the master Keeper is holding the NR for. Since all information regarding the NR is contained in the TUI Lpacket being broadcast by the master Keeper, and since backup Keepers are receiving the broadcast TUI and are using the broadcast TUI for determining if and when they should take over as master Keeper, the backup Keepers shall have all the information needed to cleanly transition holding the NR when the appropriate backup Keeper takes over as master Keeper.

Faulted backup and master Keeper shall process Network Resource requests in the same way as non-faulted backup and master Keeper.

When the network resource is not being held, the related fields in the network TUI Lpackets shall be zeroed:

- NR_VENDOR_ID;
- NR_SERIAL_NUMBER;
- NR_CLASS;
- NR_INSTANCE.

7.3.6.3 Obtain_Network_Resource service

The Obtain_Network_Resource service request shall cause the master Keeper to hold the Network Resource for 60 s for itself or another node, if it is not already being held. The time period that the Network Resource is held may be extended indefinitely through use of the Hold_Network_Resource service request.

If the Network Resource is already being held for any node, an error status shall be returned.

The Obtain_Network_Resource service shall return the error codes as defined in Table 29, as designated in column ONR for this service.

7.3.6.4 Hold_Network_Resource service

The Hold_Network_Resource service request shall cause the master Keeper to continue holding the Network Resource for the next 60 s if the node requesting the hold is the same node for which the master Keeper is currently holding the Network Resource.

If the Network Resource is being held for another node or is not being held, an error status shall be returned.

The Hold_Network_Resource service shall return the error codes as defined in Table 29, as designated in column HNR for this service.

7.3.6.5 Release_Network_Resource service

The Release_Network_Resource service request shall cause the master Keeper to stop holding the Network Resource if the node requesting the release is the same node for which the master Keeper is currently holding the Network Resource. If a network change is in process and the network resource is released, the network change will be aborted.

Whether the Network Resource is being held for another node or is not being held, an error status shall be returned.

The Release_Network_Resource service shall return the error codes as described in Table 29, as designated in column RNR for this service.

7.3.6.6 Change_Start service

The Change_Start service request shall cause the Keeper object to:

- a) copy the current copy of the attributes in non-volatile storage into the pending copy in RAM;
- b) clear the count of Set_Attribute service requests (including set single, list and fragment) received by the ControlNet object;
- c) put the Keeper object into the appropriate net_change operating state.

This service request shall only be processed if the Network Resource is being held by the master Keeper as indicated by the net_resource bit in the broadcast TUI Lpacket.

The Change_Start service shall return the error codes as defined in Table 29, as designated in column CS for this service.

7.3.6.7 Change_Complete service

The Change_Complete service request shall cause the Keeper object to:

- a) copy its pending copy of the attributes from RAM to the current copy of the attributes in non-volatile storage;
- b) perform a synchronized network change;
- c) verify tool_keeper_revision in attribute 105 is $\leq$ Keeper object revision (if not true, mark attribute table invalid);
- d) exit whatever network change operating state it is in;
- e) return to normal Keeper operation.

If the Keeper object is not in one of the net change operating states, the Change_Complete service request shall be ignored and an incorrect state error response shall be generated.

If the number of Set_Attribute service requests (including: Set_Attribute_Single, Set_Attribute_List and Set_Attribute_Fragment) received by the Keeper object since the Change_Start service request does not match the number in the Change_complete parameter, a Change_Abort shall be executed instead and an error response shall be generated.

The reply to this service shall be returned after at least one TUI is broadcast with the net_change_in_progress bit cleared. A synchronized change shall take place prior to the TUI transmission with the bit reset.

The Change_Complete service shall return the error codes as defined in Table 29, as designated in column CC for this service.

7.3.6.8 Change_Abort service

The Change_Abort service shall cause the Keeper object to

- a) discard changes made to the pending attribute data;
- b) release the network resource;
- c) return to normal Keeper operation.

The reply to this service shall be returned after at least one TUI is broadcast with the net_change_in_progress bit and holding_network_resource bit cleared.

The Change_Abort service shall return the error codes as defined in Table 29, as designated in column CA for this service.

7.3.6.9 Get_Signature service

The Get_Signature service shall cause the master Keeper to look up the CO password values for a connection originator in the CO_data attribute. The path to the device shall be given as the attribute in the request. This path shall be parsed and used to search the tree structure of the CO_data one branch at a time, to locate the appropriate COP (CO password) value, which is returned as a parameter in the response.

The path shall be parsed one branch at a time since the path entries in the CO_data attribute only specify the portion of the path between one branch and the next, not the entire path to that branch.

The Get_Signature service shall return the error codes as specified in Table 29, as designated in column GS for this service.

7.3.6.10 Get_Attribute_Fragment service/response

The Get_Attribute_Fragment service shall collect and return some portion of the CO_data attribute data managed by the Keeper object. Any part of this attribute may be read. The CO_data attribute shall be the only Keeper attribute which responds to fragment services. Accessing other attributes shall cause an error to be returned.

NOTE When getting attribute data, the current attribute data from non-volatile storage, not the pending copy of the attribute should be returned.

The Get_Attribute_Fragment service shall return the error codes as defined in Table 29, as designated in column GAF for this service.

7.3.6.11 Set_Attribute_Fragment service

The Set_Attribute_Fragment service shall be used to set any portion of the CO_data attribute managed by the Keeper object. Only the CO_data attribute shall be accessed via this service. Accessing other attributes shall cause an error to be returned.

The Set_Attribute_Fragment service shall be processed only when the Keeper object is in one of the net change operating states. The Keeper object shall return an invalid mode error if it is not in one of the net change operating states when a Set_Attribute_Fragment service request is received.

Setting attribute data, shall set the pending copy of the attribute, not the current copy in non-volatile storage. The pending attributes shall be copied to the current attributes in non-volatile storage upon reception of a class specific Change_Complete service request.

The Keeper object shall not perform attribute value checking during any set operation.

The Set_Attribute_Fragment service shall return the error codes as defined in Table 29, as designated in column SAF for this service.

7.3.6.12 Table unique identifier (broadcast), fixed tag 0x84

When the Keeper object is in the master verify, faulted_master_verify, master, faulted_master, net change master or net_change_faulted_master operating state, it shall attempt to transmit a special Table Unique Identifier (TUI) Lpacket as scheduled data once every 4 NUTs (that is, on NUT numbers 0, 4, 8, 12, 16 ...). All Keeper devices shall reserve 26 octets of scheduled link transmit time once every 4 NUTs for transmitting this Lpacket. If sent unscheduled, it shall be the highest QoS priority unscheduled information.

When generating the transmitted TUI Lpacket, any reserved fields shall use the values as specified in the corresponding reserved fields of attribute 0x0101. The programming tool shall set any reserved fields in attribute 0x0101, to zero.

The format of the TUI Lpacket link data shall be as defined in Table 30.

Table 30 – Wire order format of the TUI Lpacket

Name	Data type	Description of parameter	Semantics of values
Size	USINT	Size of the Lpacket in words.	0x0D
Control	USINT	Link layer Lpacket control octet.	0x01 (Fixed tag Lpacket)
Fixed_Tag	USINT	Fixed tag value.	0x84 (TUI Lpacket)
Destination_Mac_Id	USINT	Who receives the Lpacket.	0xFF (Broadcast)
Unique_Id	UDINT	CRC of the Keeper's current attributes.	Least significant octet first.
Status_Flag	UINT	TUI flag values.	See the TUI attribute for details.
Keeper_Mac_Id	USINT	MAC ID of the Keeper broadcasting the TUI.	See the TUI attribute for details.
Reserved	USINT	Reserved for data alignment	
Net_Resource_Vendor_Id	UINT	Vendor ID of object holding Net Resource for exclusive use	
Net_Resource_Serial_Number	UDINT	Serial number of object holding Net Resource for exclusive use	
Net_Resource_Class	UDINT	Class number of object holding Net Resource for exclusive use	
Net_Resource_Instance	UDINT	Instance number of object holding Net Resource for exclusive use	

7.3.7 Service error codes

Table 31 lists possible error codes and the most likely condition under which the code would be returned.

Table 31 – Service error codes

Error code	Meaning	Return condition
0x02	resource_unavailable_err	The requested network resource is not available
0x03	invalid_parameter_err	An invalid parameter was provided to the service
0x04	path_segment_err	Whenever an path over and above the attribute number is specified
0x05	path_dest_unknown_err	Object instance does not exist
0x08	unimplemented_service_err	Whenever the service is not supported for an attribute
0x09	invalid_attr_err	Whenever an attribute is specified which is not supported in this object
0x0C	wrong_object_state_err	The object is in a state which does not allow the service to be performed
0x0D	already_exists_err	The service has already been obtained by the requesting node
0x0E	not_settable_err	Attribute is not settable
0x0F	no_permission_err	Node requesting this service does not currently own the resource
0x11	reply_too_large_err	Not enough room in the response buffer to reply
0x13	not_enough_data_err	Whenever the request does not contain enough data
0x14	undefined_attr_err	Attempts to access an undefined attribute
0x15	too_much_data_err	More data has been encountered than expected for this service request
0x16	nonexistent_object_err	Keeper object not available
0x26	invalid_path_size_err	Whenever an invalid path segment for this service is specified

7.3.8 Behavior

The numeric indication of Keeper state shall be as defined in Table 32.

Table 32 – Keeper object operating states

State	Description
0,1,2 – Power up	The Keeper object is either waiting for the node to come on-line or is determining if it is a legitimate Keeper for this link
3 – Backup	The Keeper object has determined that it is a legitimate Keeper for this link but either has determined it is a backup Keeper or has not yet determined if it is the master Keeper for this link
4 – Master Verify	The Keeper object has determined that it is a legitimate Keeper for this link but has not heard another active Keeper on the link or has received a good TUI from a matching Keeper at a higher node number or has heard a TUI from a faulted Keeper. It starts broadcasting the TUI Lpacket but does not process synchronized parameter changes or respond to requests of the master Keeper
5 – Master	The Keeper object has determined that it is a legitimate Keeper for this link and that it is the master Keeper for this link. It broadcasts the TUI Lpacket, processes synchronized link changes when required to change link parameters and shall respond to requests of the master Keeper
6 – Faulted Backup	The Keeper object has determined that it is not a legitimate Keeper for this link, and is not the Keeper if there are only faulted Keepers on the link
7 – Faulted Master Verify	The Keeper object is faulted but has not heard another active Keeper on the network for 12 NUT times. It starts broadcasting the TUI Lpacket but does not process synchronized link changes or respond to requests of the master Keeper
8 – Faulted Master	The Keeper object is faulted and has assumed faulted master Keeper duties. There are no unfaulted Keeper's on the network. It broadcasts the TUI Lpacket, processes synchronized link changes when required to change network parameters and shall respond to requests of the master Keeper
9,10,11,12 – Net Change	The Keeper is having its attributes updated. There is a <i>Net Change</i> substate for the <i>backup</i> , <i>master</i> , <i>fault</i> and <i>faulted master</i> states previously discussed. The separate substates are needed so the Keeper object can exit the <i>Net Change</i> state properly when updates complete normally and when updates are aborted. The pending attributes may be set only when the Keeper is in a <i>Net Change</i> state. Nodes in the <i>net change master</i> and <i>net change faulted master</i> shall respond to requests of the master Keeper. Nodes in the <i>net change backup</i> or <i>net change faulted backup</i> states shall not respond to requests of the master Keeper

7.3.9 Miscellaneous notes

Keepers shall maintain parameters only for nodes on the same link.

Each Keeper shall be capable of distributing link parameters to all nodes on that link. The master Keeper shall be the Keeper with the lowest MAC ID of all the legitimate Keepers on the link. A faulted master Keeper on a link may or may not be the lowest MAC ID.

The Keeper shall be responsible for configuring those devices that appear in its attributes. The master Keeper shall be the only Keeper that issues responses to service requests via fixed tag 0x88 that are broadcast to all Keeper objects on a net.

All legitimate Keeper objects on a link shall have identical copies of the network attributes for all devices on the link. Keepers that do not agree with the master Keeper shall stay in the faulted backup state until they agree (agree means their TUI *unique_id* parameters are identical).

Only one programming terminal may modify Keeper attributes at a time. The programming terminal shall acquire the Network Resource prior to starting the Keeper modification and shall retain possession of the Network Resource during the entire modification procedure. If a programming terminal makes a change to a Keeper, it shall update all of the other Keepers on the link as well by broadcasting the Change_Start, Set_Attribute, and Change_Stop service requests.

The Keeper shall check the “tool Keeper revision” in attribute 0x0104 for a valid attribute table. If the Keeper is unable to understand that revision, the Keeper shall clear bit 5 in any TUI transmissions so that other more capable Keepers can assume the master Keeper responsibilities. Keeper revisions 0 and 1 shall be identical to revision 2.

The “connection info revision” value is independent on the Keeper revision, and shall only be utilized by programming tools.

The Keeper may, for network diagnostic purposes, broadcast to all nodes a debug fixed tag (0x90) Lpacket with three words of data. The first word identifies that the Keeper object is sending the diagnostic. The second word is the current “Keeper state” and the third word is the next “Keeper state”.

7.3.10 Keeper power up sequence

7.3.10.1 General

The Keeper power up sequence shall allow the Keeper object to begin operations under a variety of conditions, from a normal configured network power on state to one where many network devices are new and unconfigured.

The Keeper object power up sequence shall not begin until the NAM has brought the Keeper node on-line. The Keeper power up shall determine if this particular Keeper device is a legitimate Keeper for the local link and to bring the Keeper object into either the backup or faulted backup operating state.

If this device is a legitimate Keeper for the link, the Keeper object shall determine if it is the master Keeper or a backup Keeper for the link. If this device is not a legitimate Keeper for the link, the Keeper object enters a fault state. An unconfigured Keeper shall enter the fault state at power up.

The 12*NUT timer shall be retriggered whenever a TUI Lpacket is received that indicates the NR is being held. This shall indicate that the Keeper attributes are being updated.

The Keeper object's power up sequence is illustrated in Figure 18.

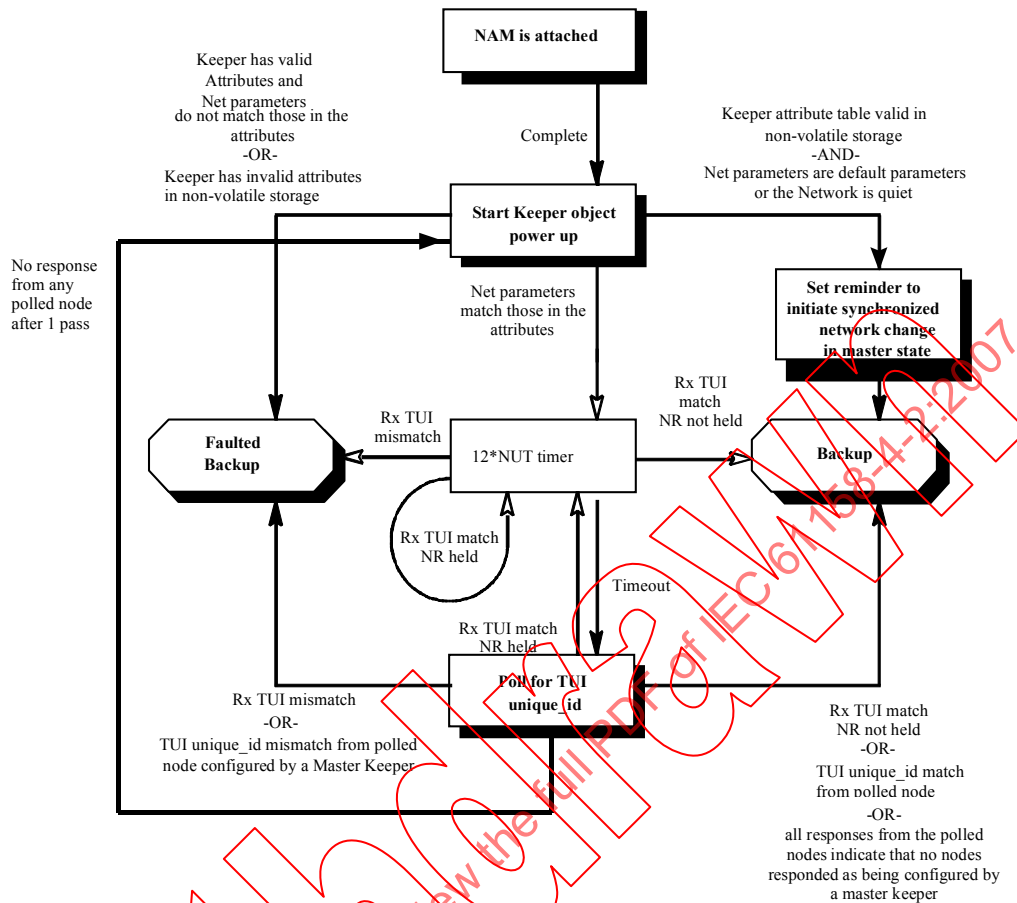


Figure 18 – Keeper object power-up state diagram

7.3.10.2 Poll for TUI unique_id

The Poll for TUI unique_id state in Figure 18 shall require the following operation be performed by the Keeper object:

- Issue a UCMM request to the addressed node's ControlNet object to read the object's current configuration parameters. Should the request fail, proceed on to the next MAC ID until UMAX is reached. Multiple concurrent requests may be permitted to reduce the poll time.
- Each MAC ID in the range 1 through UMAX (excluding the MAC ID of the Keeper node) may be polled singly or concurrently with a Get_Attribute_Single request on the current_link_config attribute of the ControlNet object of the node. This attribute contains a copy of the current TUI attribute for the node. Concurrent access to multiple nodes at one instant will serve to accelerate the polling process.
- If no response is heard from any polled node and UMAX has been reached, the poll is aborted and the Keeper object shall transition to the "Start Keeper object power up state", since at least one other node on a non-default network is present but unable to respond. This will allow the power up poll process to repeat until a node is capable of responding.
- If a response is received from a polled node and the "Keeper State" bit in the returned TUI status_flag was set, the polled node was configured by a Keeper in the "Master" operating state. Compare the TUI unique_id value returned by the polled node to the one in this Keeper's attribute table. If it matches, the poll is aborted and the Keeper object shall transition to the "backup" operating state. If it mismatches, the poll is aborted and the Keeper object shall transition to the "faulted backup" operating state.

- e) If a response is received from a polled node and the “Keeper State” bit in the returned TUI status_flag was not set, the polled node was not configured by a Keeper in the “Master” operating state. In this case, the information is discarded and the poll continued. If all nodes respond in this way, indicating that none of them were configured by a Keeper in the “Master” operating state, the poll is terminated and the Keeper shall transition to the “backup” operating state since in this situation, all nodes are either new or have lost their stored configuration.

7.3.10.3 Operating states

The Keeper object operating states shall be defined as in Figure 19, Table 33, and the following sections.

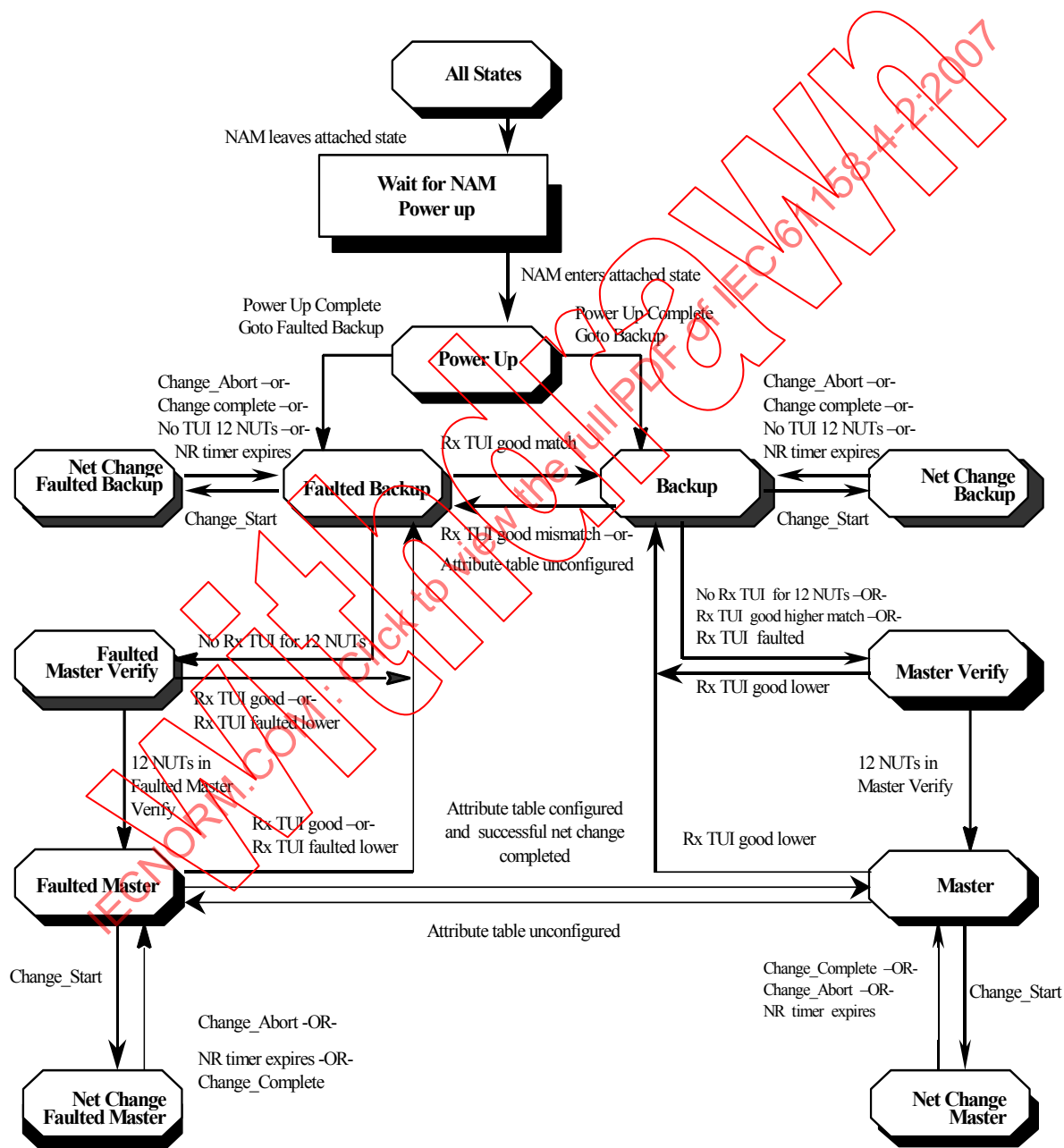


Figure 19 – Keeper object operating state diagram

Table 33 – Keeper object state event matrix

Event	State										
	Power Up	Backup	Master Verify	Master	Faulted Backup	Faulted Master Verify	Faulted Master	Net Change (Back-up)	Net Change (Master)	Net Change (Faulted Backup)	Net Change (Faulted Master)
Rx TUI from a good node						Goto Faulted Backup	Goto Faulted Backup				
Rx TUI from a good lower node			Goto Backup								
Rx TUI from a good higher node (match)		Goto Master verify									
Rx TUI from a good node (mismatch)		Goto Faulted Backup									
Rx TUI from a good node (match)					Goto Backup						
Rx TUI from a faulted node		Goto Master Verify									
Rx TUI from a faulted lower node						Goto Faulted Backup					
No TUI Rx for 12*NUT		Goto Master Verify			Goto Faulted Master Verify			Abort change. Goto Backup		Abort change. Goto Faulted Backup	
12 Nut times elapse			Goto Master			Goto Faulted Master					
Change_Start		Goto Net Change Backup		Goto Net Change Master	Goto Net Change Faulted Backup		Goto Net Change Faulted Master				
Change_Complete								Goto Backup	Goto Master	Goto Backup	Goto Master verify
Change_Abort								Do Change Abort. Goto Backup	Do Change Abort. Goto Master	Do Change Abort. Goto Faulted Backup	Do Change Abort. Goto Faulted Master
60 s Network Resource timer expires								Abort change. Goto Backup	Abort change. Goto Master	Abort change. Goto Faulted Backup	Abort change. Goto Faulted Master
NAM leaves attached state	Goto wait for NAM powerup										
Power up complete to Backup	Goto Backup										
Power up complete to Faulted Backup	Goto Faulted Backup										

NOTE Blank cells indicate that this event cannot occur, or if it does occur, no action shall be taken by the Keeper object.

7.3.10.4 Network resource

Access to changing Keepers shall be controlled through the use of the Network Resource. Only one device can acquire the Network Resource for exclusive access at a time.

The Network Resource shall be held by the master Keeper for the node performing the attribute updates. If the master Keeper does not hear from the node performing the update at least once every 60 s, the master Keeper shall automatically release the Network Resource, abort the change in progress, and return to normal operation.

7.3.10.5 Faulted Keeper

Any Keeper whose *TUI unique_id* attribute does not match that of the broadcast TUI from a Keeper shall enter a fault state unless a net change operation is in progress.

7.3.10.6 MAC ID chooses master Keeper

The Keeper node MAC ID included in the TUI message shall be used by the Keeper to determine which Keeper on a link is the master Keeper.

7.3.10.7 Master verify

The Keeper shall wait for $12 \times \text{NUT}$ after the first TUI broadcast begins before assuming master Keeper status. This shall be sufficient time for a Keeper with a lower MAC ID to become master Keeper. Other Keepers become backup Keepers.

7.3.10.8 Rogue

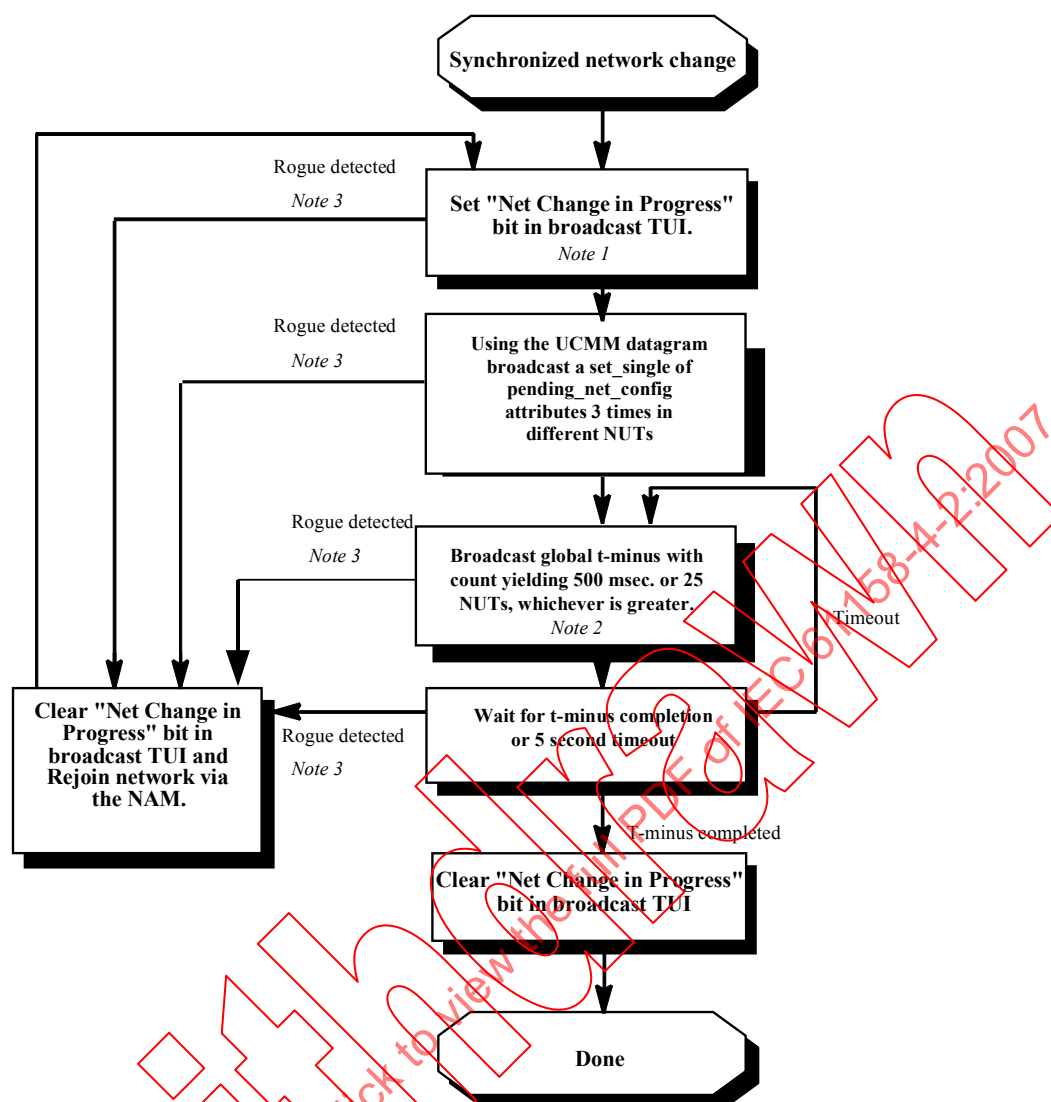
If the node containing the Keeper rogues during a synchronized network change sequence, the "Net Change in Progress" bit shall be cleared and control shall be returned back to the start of the synchronized network change processing immediately after the NAM brings the node back on-line. The normal Keeper object power-up sequence shall be bypassed.

7.3.10.9 Synchronized network change processing algorithm

Although the moderator is used by the link to distribute operating parameters, it shall be the responsibility of the Keeper object to initiate changes to the link parameters.

All link parameter changes shall be accomplished through the use of a synchronized network change algorithm. This shall provide a means for all nodes on the network to change their current link parameters in unison without disrupting network operation.

The algorithm for the synchronized network change operation shall be as defined in Figure 20.



NOTE 1 Broadcasting a TUI with the net change in progress bit set forces nodes that are just powering up to hold off from going on-line until after the network change completes and the net change in progress bit in the broadcast TUI is cleared. This reduces the chances of a rogue event occurring.

NOTE 2 The tminus delay count provides noise immunity and insures that all nodes will hear at least one moderator with the new tminus count. It also allows nodes a reasonable amount of time to process the Set_Single packets with new network attributes. The delay count is determined by taking the maximum of 25 or the result of dividing 500 ms by the current nut time in ms.

NOTE 3 The rogue condition will occur only if the moderator node and at least one other node did not receive one of the three broadcast set network attribute service request packets. This would only occur if there was a serious noise event (lasting at least 3 NUTs in duration).

Figure 20 – Synchronized network change processing

7.4 Scheduling object

7.4.1 Overview

The Scheduling object shall exist in connection originator (CO) devices. The Scheduling object shall be used by link scheduling software (LSS) to

- read scheduled connection information;
- write schedule information;
- provide the CO with a password (software key) which shall be used to authenticate the CO device's access to a specific link.

A LSS session with the Scheduling object schedules a particular link. If a CO device has connections on multiple links, multiple LSS sessions shall be needed to schedule all the connections. The results of the scheduling session shall be saved by the CO device (including the CO password computed by the LSS).

Since the LSS may be integrated with the programming software (PS) for a CO device, the Scheduling object shall provide commands which support integrated or a separate PS. Specifically, when the LSS writes Scheduling data to the Scheduling object, it may write conditionally or it may force the write. A conditional write of Scheduling data shall be used if the PS and LSS were not integrated and the Scheduling object shall only use the provided Scheduling data if a separate PS has not changed the connection information during the LSS session. A forced write may be used with an integrated LSS/PS if the integrated tool can guarantee that no other PS has changed connection information since it was last read during the current Scheduling session.

All communication to the Scheduling object shall use the Unconnected Message Manager (UCMM) fixed service, or an active Generic tag connection service to the Message Router object.

7.4.2 Class attributes

The Scheduling object shall support the class attributes as specified in Table 34. The Get_Attribute_All service shall return the class level attributes in numerical order, smallest to largest.

Table 34 – Scheduling object class attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute
0x01	Required	Get	Revision	UINT	First revision, value = 1
0x02	Required	Get	MaxInstances	UDINT	maximum number of Scheduling objects supported
0x03	Required	Get	NumInstances	UDINT	number of instantiated Scheduling objects

7.4.3 Instance attributes

The Scheduling object shall support the instance attributes as specified in Table 35. The Get_Attribute_All service shall return the attribute level attributes in numerical order, smallest to largest.

Table 35 – Scheduling object instance attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute
0x01	Required	Get/Set	Route	STRUCT of variable size	route from CO to the link being scheduled
			NumSegments	UINT	number of segments in path
			Segments	ARRAY of UINT	array of segments
0x02	Required	Get/Set	TimeOut	UDINT	watchdog time-out μ s (default = 60 s)
0x03	Optional	Get/Set	Controller State	UINT	bit 0 (run/program) = 0, may be programmed = 1, currently controlling I/O bit 1 (remote/local) = 0, bit 0 shall not be changed remotely = 1, bit 0 may be changed remotely bit 2-15 reserved and shall be zero

7.4.4 Common services

7.4.4.1 General

In addition to the services described in this subclause, the Scheduling object shall support the standard Get_Attribute_All service addressed to the class instance (instance zero). A specific Scheduling object implementation may optionally support other standard attribute services.

The Scheduling object shall support the common services as specified in Table 36.

Table 36 – Scheduling object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Required	Optional	Get_Attribute_All	Returns a predefined listing of this objects attributes Modifies all settable attributes
0x02	N/A	Optional	Set_Attribute_All	Modifies all settable attributes
0x03	Optional	Optional	Get_Attribute_List	Returns the contents of a list of specified attributes
0x04	N/A	Optional	Set_Attribute_List	Modifies a list of settable attributes
0x08	Required	Optional	Create	Used to instantiate a Scheduling object
0x09	N/A	Required	Delete	Used to conclude a LSS Scheduling session
0x0E	Optional	Optional	Get_Attribute_Single	Returns the contents of the specified attribute
0x10	N/A	Optional	Set_Attribute_Single	Modifies a single attribute

An implementation may support setting the route instance-level attribute, but if it does, the internal behavior shall be indistinguishable from a delete service followed by a create service containing the new route.

7.4.4.2 Create

The create service shall be used to instantiate a Scheduling object on the CO. Within the create message, the client (LSS) shall include a route pointing from the CO to the link to be scheduled. The route to the link shall consist of port segments and shall not contain network segments. The Scheduling object shall reply with an instance number to which all further communication shall be addressed for this Scheduling session, and with the currently stored CO password and path.

The new instance shall delete itself if communications to its client are severed. Communications to an instance shall be monitored via a watchdog timer whose time-out value (default = 60 s) can be set via one of the optional set attribute services. The watchdog timer for an instance shall be reset when it receives any message. If the timer expires, the instance shall be deleted aborting all pending changes. The time-out value shall be a 32-bit integer with units of μ s.

If the create service is addressed to the class level (instance zero), the instance number of the newly created instance shall be chosen automatically by the Scheduling object. For debug purposes, the Scheduling object may optionally allow the create service to be addressed to a specific instance number.

The Scheduling object shall support at least one instance. If the Scheduling object allows more than one instance to exist, it shall ensure the integrity of all its instances. The value of instance attribute 0x01 shall be provided with the create service.

```
class Scheduling_Create_Request : public MR_Request
{
    UINT    number_of_attributes;    // set to 1
    InstanceAttribute1 attribute1;
}

class Scheduling_Create_Response : public MR_Response
{
    UDINT   instance_number;
    UDINT   cop;                    // Connection Originator Password (COP)
    UINT    path_size;              // in words, range 0 to 255
    UINT    path[];                 // array of port segments from the scheduled link
                                     // to the connection originator
                                     // no key segments, no network segments
                                     // include the MAC ID of the link hop
};
```

The response shall include one of the status codes shown in Table 37.

Table 37 – Status error descriptions for Create

General status	Extended status	Error description
0x02	0x0001	Insufficient resource — maximum number of Scheduling object instances already exist
	0x0002	Insufficient resource — not enough memory on CO device
0x04	N/A	Path syntax error
0x08	N/A	Unimplemented service (since non-zero instances are optional)
0x10	N/A	Cannot open another instance due to internal conflicts
0x13	N/A	Insufficient request data — request was too short or truncated
0x0D	N/A	Object already exists (when non-zero instance specified)
0x1C	N/A	Attribute list shortage — required attribute was missing
0xD0	N/A	No scheduled connections on this link

7.4.4.3 Delete

The delete service shall be used to conclude a LSS Scheduling session. It deallocates all resources for a specified instance of the Scheduling object. Any pending changes which have not yet been committed shall be lost. Connections which have been broken by commands earlier in the LSS session shall not necessarily be restored. The delete service shall not be supported at the class level. The response of this service shall be as specified in Table 38.

Table 38 – Status error descriptions for Delete and Kick_Timer

General status	Extended status	Error description
0x00	N/A	Success
0x04	N/A	Path syntax error
0x05	N/A	Instance undefined
0x08	N/A	Unimplemented service (if directed to instance 0)

7.4.5 Class specific services

7.4.5.1 General

The Scheduling object shall support the following class specific services as specified in Table 39.

Table 39 – Scheduling object class specific services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x4B	N/A	Required	Kick_Timer	Reset the watchdog timer within the Scheduling object
0x4C	N/A	Required	Read	Read information about the scheduled connections
0x4D	N/A	Required	Conditional_Write	Write the schedule information back to the CO device
0x4E	N/A	Optional	Forced Write	Write the schedule information back to the CO device
0x4F	N/A	Required	Change Start	Initiate a Scheduling data change
0x50	N/A	Required	Break Connections	Break all affected connections after new Scheduling data
0x51	N/A	Required	Change Complete	Indicate that all scheduled connections have been broken
0x52	Required	N/A	Restart Connections	Break all connections which use a specific link

7.4.5.2 Kick_Timer

The kick_timer service shall reset the watchdog timer within an instance of the Scheduling object without otherwise affecting the state of the Scheduling object. The LSS shall issue a command at a rate of four per time-out period. For example, the LSS shall issue a service request every 15 s if the time-out value is set to the default 60 s. All other Scheduling object services, directed to this Scheduling instance, shall also reset the watchdog timer so the kick timer service need only be used if the client (LSS) wants to keep a Scheduling object instance alive but has nothing to say.

The watchdog timer for an instance shall be reset when it receives any message; however, it shall not start its countdown until the corresponding response is sent. This shall be to allow for a single task implementation. The response of this service shall be as specified in Table 38.

7.4.5.3 Read

The read service shall be used by the LSS to read information about the scheduled connections that the CO device desires to create on a specific link. The connection information shall come from an application object within the CO device.

A read service request shall contain a required UDINT parameter that specifies which connection index on which to start the read. The Scheduling object shall reply to a read service with as much connection information as can be fit into a reply Lpacket. The connection indexes within an Lpacket shall be ordered from smallest to largest (gaps shall be allowed). The extended status field within the reply shall indicate whether connections with higher indexes exist in the CO device. A client (LSS) shall continue to submit **read** requests with higher starting indexes until it reads all the connection information for the chosen link. An extended status of 0 shall indicate that more connections exist in the device. An extended status of 1 shall indicate that all connections have been read.

The reply shall consist of a UINT indicating the number of connections described in this response followed by a series of entries each describing a connection that meets the connection request criteria (the route to the link). Each entry shall have a unique Scheduling data index that is chosen by and shall primarily be used by the CO. The last one of these can be incremented by the LSS to continue the request if it is too long for a reply.

Other parameters provided in the connection entry shall describe the connection sizes, connection types (multi-cast/point to point) and connection update rates (RPIs). The connection entry route information shall consist of the route from the link to the target module for the connection including the connection points within the target module. Port information need not actually be used by the LSS.

This route shall also contain the current values assigned for the connection in the network segments. The network segments shall contain both API (Lpacket interval) and starting NUT information.

```
class Scheduling_Read_Request : public MR_Request
{
    UDINT first_connection_index;
}

class Scheduling_Read_Response : public MR_Response
{
    UINT number_of_connections;
    Scheduling_Connection_Description desc[];
}

class Scheduling_Connection_Description
{
    UDINT connection_index;
    UINT O2T_parameters;
    UINT T2O_parameters;
    UDINT O2T_RPI;
    UDINT T2O_RPI;
    UINT path_size;           // in words, range 0 to 255
    UINT path[];             // array of path segments
                                // first segment is always O=>T schedule segment
                                // second is always port segment onto the link
                                // third is always T=>O schedule segment
                                // remaining path including logical segments
                                // and other port segments if multihop connection
};
```

The scheduling tool, LSS, shall use the logical segments of the target path to determine whether more than one connection request actually uses one multipoint producer. The rule for all objects except the Assembly object shall be that a match of class, instance, T=>O connection point, and any finer granularity (for example attribute or member) constitutes a unique producer and it shall be scheduled once for both connections. Since the Assembly

object equates instance and connection point, a match on connection path instance shall not determine whether the producer is unique.

For example, these two connection paths specify the same producer (instance 0x88 of the Assembly object):

"20 04 24 03 2C 99 2C 88";

"20 04 24 AA 2C 77 2C 88".

These two connection paths specify different producers since the logical class segment do not match:

"20 77 24 03 2C 99 2C 88";

"20 77 24 AA 2C 99 2C 88".

The status response of this service shall be as specified in Table 40.

Table 40 – Status error descriptions for Read

General status	Extended status	Error description
0x00	N/A	Success
0x04	N/A	Path syntax error
0x05	N/A	Instance undefined
0x08	N/A	Unimplemented service (if directed to instance 0)
0x13	N/A	Insufficient request data — request was too short or truncated

7.4.5.4 Conditional_Write

Conditional_Write service shall be used by the LSS to write the schedule information back to the CO device. Receipt of this message shall indicate that specific indexes of the connection information are being rescheduled by the LSS. A CO device may start breaking the current connections associated with the connection indexes, or it may wait until the receipt of a subsequent break connections service request.

If the new schedule provided by the LSS exactly matches the current schedule for a given connection, the CO device may ignore that part of the write request. A Scheduling object implementation may reply with an out of resource error if it receives updates for more connections than it has been queried about (via read requests). An implementation may also check for duplicate write requests; thereby, allowing retries — more write requests than read requests. The connection indexes may not be in order.

The difference between the conditional write service and the forced write service shall be that, for the conditional case, the CO device shall be required to verify the freshness of the connection information it previously sent to the LSS. If the data is not fresh, the CO device shall not use the Scheduling data for that connection index. In the forced case, the CO device need not check. "Fresh" connection information shall mean that nothing has changed the connection information since it was sent to the LSS via a read command.

The request shall be an array of connection schedules.

```
class Scheduling_Write_Request : public MR_Request
{
    UINT    number_of_connection_schedules;
    struct {
        UDINT connection_index;
        USINT O2T_schedule;
        USINT T2O_schedule;
    } schedules[];
};
```


The response status of this service shall be as specified in Table 41.

Table 41 – Status error descriptions for Conditional_Write

General status	Extended status	Error description
0x00	N/A	Success
0x03	N/A	Invalid value — bad index contained in request
0x04	N/A	Path syntax error
0x05	N/A	Instance undefined
0x08	N/A	Unimplemented service (if directed to instance 0)
0x0C	N/A	Wrong state (change start not received)
0x10	N/A	Device state conflict (device cannot break connection)
0x13	N/A	Insufficient request data — request was too short or truncated
0x15	N/A	Instance is unable to receive any more write requests

7.4.5.5 Forced_Write

Forced_Write service shall be similar to the conditional write service except that the LSS shall guarantee that it has provided valid connection information. The request parameters shall be the same as for the Conditional_Write service. The response status of this service shall be as specified in Table 42.

Table 42 – Status error descriptions for Forced_Write

General status	Extended status	Error description
0x00	N/A	Success
0x04	N/A	Path syntax error
0x05	N/A	Instance undefined
0x08	N/A	Unimplemented service (if directed to instance 0)
0x0C	N/A	Wrong state (change start not received)
0x10	N/A	Device state conflict (device cannot break connection)
0x13	N/A	Insufficient request data — request was too short or truncated
0x15	N/A	Instance is unable to receive any more write requests

7.4.5.6 Change_Start

The Change_Start service shall be used to initiate a Scheduling data change in the controller. The controller shall allocate enough memory for the new data and for the CO password and path. The **conditional write** or **forced write** service shall be then used to write the data to the controller.

The CO password shall be used when a controller first is powered up and begins to remake scheduled connections. The CO shall ensure the path associated with the CO password (see Create response or Change_Complete request) matches the path from the network to the CO device. If the path matches the CO shall ensure this CO password matches the one stored in the Keeper object of the link prior to making any scheduled connections on that link.

The Change_Start request shall have only one parameter. The parameter shall be a UINT in the range 0 to 255 that specifies the size of the CO password and path to be provided in the

Change_Complete service. Implementations may use this to allocate a receive buffer for the CO password and path.

The response status of this service shall be as specified in Table 43.

Table 43 – Status error descriptions for Change_Start

General status	Extended status	Error description
0x00	N/A	Success
0x02	0x0002	Insufficient resource — not enough memory on CO device
0x04	N/A	Path syntax error
0x05	N/A	Instance undefined
0x08	N/A	Unimplemented service (if directed to instance 0)
0x10	N/A	Device state conflict (for example, device cannot allocate sufficient memory)
0x13	N/A	Insufficient request data — request was too short or truncated

7.4.5.7 Break_Connections

Break_Connections service shall be used after the new Scheduling data has been written to the controller to break all the affected connections. The service shall reply only after all the connections have been broken. If any connections are in the process of being connected, they shall also be closed before replying to this service.

The Scheduling object break connections service shall break all connections that were specified in the Scheduling data write services. An LSS may always write Scheduling data for all connections (causing all connections over the link to be broken) but another LSS may be optimized to minimize the number of connections that get broken.

When the Scheduling object break connections service is received, in addition to breaking the changed connections, the CO shall NULL out (zero out) the currently active scheduled data values for the connections it breaks to prevent connections from being re-established until the Scheduling data change complete service is received. If the change complete service is never received (due to loss of communications with LSS) these connections shall not be able to be opened until they are rescheduled. The response of this service shall be as specified in Table 44.

Table 44 – Status error descriptions for Break_Connections

General status	Extended status	Error description
0x00	N/A	Success
0x04	N/A	Path syntax error
0x05	N/A	Instance undefined
0x08	N/A	Unimplemented service (if directed to instance 0)
0x0C	N/A	Object state conflict
0x10	N/A	Device state conflict (device cannot break connections)
0x13	N/A	Insufficient request data — request was too short or truncated

7.4.5.8 Change_Complete

Change_Complete service shall be used to indicate that all scheduled connections have been broken and it is permitted to start making the new scheduled connections. The request contains the new CO password and path to be used for the link that has been scheduled. The CO shall ensure the path associated with the CO password (see Create response or Change_Complete request) matches the path from the network to the CO device. If the path matches the CO shall ensure this CO password matches the one stored in the Keeper object of the link prior to making any scheduled connections on that link.

```
class Scheduling_Change_Complete_Request : public MR_Request
{
    UDINT  cop;           // Connection Originator Password (COP)
    UINT   path_size;     // in words, range 0 to 255
    UINT   path[];        // array of port segments from the scheduled link
                        // to the connection originator
                        // no key segments, no network segments
                        // include the MAC ID of the link hop
};
```

NOTE The reverse path included in this request allows the LSS to detect changes in the network position of the scheduled CO during later scheduling sessions.

The response status of this service shall be as specified in Table 45.

Table 45 – Status error descriptions for Change_Complete

General status	Extended status	Error description
0x00	N/A	Success
0x03	N/A	Invalid value (new CO password and path is not formatted correctly)
0x04	N/A	Path syntax error
0x05	N/A	Instance undefined
0x08	N/A	Unimplemented service (if directed to instance 0)
0x10	N/A	Device state conflict
0x13	N/A	Insufficient request data — request was too short or truncated
0x0C	N/A	Wrong state

7.4.5.9 Restart_Connections

The Restart_Connections service may only be sent to the class instance of the Scheduling object (instance zero). It shall be used to break all connections which use a specific link. After all the connections have been broken, the CO device shall attempt to re-establish them. The request parameter for this service shall be structured as is instance attribute 0x01. The response status of this service shall be as specified in Table 46.

Table 46 – Status error descriptions for Restart_Connections

General status	Extended status	Error description
0x00	N/A	Success
0x04	N/A	Path syntax error
0x08	N/A	Unimplemented service (if addressed to a non-zero instance)
0x10	N/A	Device state conflict (cannot break connections)
0x13	N/A	Insufficient request data — request was too short or truncated
0xD0	N/A	No scheduled connections on this link

7.4.6 Typical scheduling session

A typical scheduling session shall use these services:

- a) create;
- b) read;
- c) write;
- d) change_start;
- e) break_connections;
- f) conditional_write;
- g) forced_write;
- h) change_complete.

The following steps describe the usage of services in a typical Scheduling session:

- 1) The LSS shall begin by sending a **create** message to instance zero of the Scheduling object. Contained in this message shall be a path from the CO to the link. This path shall determine which connections are of interest to the LSS.
- 2) The Scheduling object shall create an instance of itself to service this LSS and reply with the instance number to which all future communications in this session shall be sent.
- 3) The newly created instance shall reset a 60 s timer upon receipt of any message. If this timer ever expires, the instance shall delete itself aborting any change in progress. In normal operation, the LSS shall be required to send a command to the Scheduling object at least once every 15 s allowing a few dropped messages without unnecessarily deleting an instance.
- 4) The LSS may optionally send **read** commands to the newly created instance to transfer the connection information from the CO device. The transmitted connection information shall contain indexes which are used to uniquely identify the connections internally within the CO device.
- 5) The LSS shall reflect back these indexes on subsequent **write** commands so that the CO may determine for which connections the **write** applies. The issuing of **read** commands shall be optional since the connection information can be obtained for the LSS by other means. For example, in an integrated LSS/PS, the PS shall provide this information to the LSS.
- 6) Once the connection information is acquired from each CO desiring scheduled data on the link, the LSS shall calculate a tentative schedule.
- 7) The LSS then shall issue the following sequence of commands to each of the CO devices: **change start**, one or more **writes**, and **break connections**. This sequence can be done in parallel to each of the CO devices. The type of **write** command (forced or conditional) shall depend on how the connection information was obtained. If the LSS has independently ensured that the connection information is valid, it may use the **forced write** command; otherwise, it shall use a **conditional write** command. The LSS can know

that the connection information is valid by obtaining the edit resource for the CO before the connection information.

- 8) Upon receipt of a **conditional write** command, the Scheduling object shall check that the parameters of the connection have not changed since they were last read. If the connection has been changed, the Scheduling object shall not use the Scheduling information for the index whose connection parameters have changed. The Scheduling object within the CO can do this through any means it chooses.
- 9) The Scheduling object shall be absolved of responsibility to check the validity of **forced writes**. This responsibility shall be assumed by the LSS.
- 10) Once the **break connection** commands to each of the CO devices has completed successfully, the LSS shall issue a **change complete** to each of the CO devices indicating that the schedule transmitted earlier via **writes** is now valid. The **change complete** command shall include the new CO password and path.
- 11) The session shall end by deleting the instance of the Scheduling object.

7.5 TCP/IP Interface object

7.5.1 Overview

The TCP/IP Interface object provides the mechanism to configure the TCP/IP interface of a device.

NOTE 1 Examples of configurable items include the device's IP Address, Network Mask, and Gateway Address.

The physical port associated with the TCP/IP Interface object shall be any port supporting the TCP/IP protocol.

NOTE 2 For example, a TCP/IP Interface object may be associated any of the following : an Ethernet ISO/IEC 8802-3 port, an ATM port, a serial port running SLIP, a serial port running PPP.

The TCP/IP Interface object provides an attribute that identifies the link-specific object for the associated physical port. The link-specific object is generally expected to provide link-specific counters as well as any link-specific configuration attributes.

Each device shall support exactly one instance of the TCP/IP Interface object for each TCP/IP-capable port on the device. A request to access instance 1 of the TCP/IP Interface object shall always refer to the instance associated with the port over which the request was received.

7.5.2 Class attributes

The TCP/IP Interface object shall support the class attributes as specified in Table 47.

Table 47 – TCP/IP Interface object class attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x01	Optional	Get	Revision	UINT	revision of this object	first revision, value = 1

7.5.3 Instance attributes

7.5.3.1 General

The TCP/IP Interface object shall support the instance attributes as specified in Table 48.

Table 48 – TCP/IP Interface object instance attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x01	Required	Get	Status	DWORD	Interface status	See 7.5.3.2
0x02	Required	Get	Configuration capability	DWORD	Interface capability flags	Bit map of capability flags. See 7.5.3.3
0x03	Required	Get/Set	Configuration control	DWORD	Interface control flags	Bit map of control flags. See 7.5.3.4
0x04	Required	Get	Physical link object	STRUCT of variable size	Path to physical link object	See 7.5.3.5
			Path size	UINT	Size of path	Number of UINTs in Path
			Path	ARRAY of UINT	Logical segments identifying the physical link object	Class Segment and Instance Segment. Maximum length is 6 UINTs (if 32 bit logical segments are used)
0x05	Required	Get/Set	Interface configuration	STRUCT of variable size	TCP/IP network interface configuration	See 7.5.3.6
			IP address	UDINT	Device's IP address	Value of 0 indicates no IP address has been configured. Otherwise, the IP address shall be set to a valid class A, B, or C address and shall not be set to the loopback address (127.0.0.1)
			Network mask	UDINT	Device's network mask	Value of 0 indicates no network mask address has been configured
			Gateway address	UDINT	gateway address	Value of 0 indicates no IP address has been configured. Otherwise, the IP address shall be set to a valid class A, B, or C address and shall not be set to the loopback address (127.0.0.1)
			Name server	UDINT	Primary name server	Value of 0 indicates no name server address has been configured. Otherwise, the name server address shall be set to a valid class A, B, or C address
			Name server 2	UDINT	Secondary name server	Value of 0 indicates no secondary name server address has been configured. Otherwise, the name server address shall be set to a valid class A, B, or C address

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
			Domain name	STRING	Default domain name	ASCII characters. Maximum length is 48 characters. Shall be padded to an even number of characters (pad not included in length). A length of 0 shall indicate no Domain Name is configured
0x06	Required	Get Set is optional (see 7.5.3.7)	Host name	STRING	Host name	ASCII characters. Maximum length is 64 characters. Shall be padded to an even number of characters (pad not included in length). A length of 0 shall indicate no Host Name is configured. See 7.5.3.7
0x07	Conditional ^a		Safety network number	OCTET[6]		See IEC 61784-3-2
0x08	Conditional ^b	Get Set is conditional ^c	TTL value	USINT	TTL value for CP 2/2 multicast packets	Time-to-live value for IP multicast packets. Default value is 1, minimum is 1, maximum is 255. See 7.5.3.8
0x09	Conditional ^b	Get Set is conditional ^c	Mcast config	STRUCT of 8 octets	IP multicast address configuration	See 7.5.3.9
			Alloc control	USINT	Multicast address allocation control word. Determines how addresses are allocated	Determines whether multicast addresses are generated via algorithm or are explicitly set. See 7.5.3.9 for details
			Reserved	USINT	Reserved for future use	Shall be 0
			Num mcast	UINT	Number of IP multicast addresses to allocate for CP 2/2	The number of IP multicast addresses allocated, starting at "Mcast start addr". Maximum value is device specific, however shall not exceed the number of CP 2/2 multicast connections supported by the device
			Mcast start addr	UDINT	Starting multicast address from which to begin allocation	IP multicast address (Class D). A block of "Num mcast" addresses is allocated starting with this address

^a This attribute is required for CPF 2 safety devices. Non-safety devices shall not implement this attribute.

^b If either "TTL value" or "Mcast config" is implemented, both shall be implemented.

^c If either "TTL value" or "Mcast config" is implemented as settable, both shall be implemented as settable.

7.5.3.2 Status

The status attribute is a bitmap that indicates the status of the TCP/IP network interface, as specified in Table 49. Refer to the state diagram in Figure 21 for a description of object states as they relate to the status attribute.

Table 49 – Status bits

Bit(s)	Name	Definition
0-3	Interface configuration status	Indicates the status of the interface configuration attribute. 0 = The interface configuration attribute has not been configured 1 = The interface configuration attribute contains valid configuration obtained from BOOTP, DHCP or non-volatile storage 2 = The interface configuration attribute contains valid configuration, obtained from hardware settings (e.g. pushwheel, thumbwheel,...) 3-15 = Reserved for future use
4	Mcast pending	Indicates a pending configuration change in the TTL value and/or Mcast config attributes. This bit shall be set when either the TTL value or Mcast config attribute is set, and shall be cleared the next time the device starts
5-31	Reserved	Reserved for future use and shall be set to zero

7.5.3.3 Configuration capability

The configuration capability attribute is a bitmap that indicates the device's support for optional network configuration capability. Possible configuration capability items are listed in Table 50. Devices are not required to support any one particular item, however they shall support at least one method of obtaining an initial IP address.

Table 50 – Configuration capability bits

Bit	Name	Definition
0	BOOTP client	1 (TRUE) shall indicate the device is capable of obtaining its network configuration via BOOTP
1	DNS client	1 (TRUE) shall indicate the device is capable of resolving host names by querying a DNS server
2	DHCP client	1 (TRUE) shall indicate the device is capable of obtaining its network configuration via DHCP
3	DHCP-DNS update	1 (TRUE) shall indicate the device is capable of sending its host name in the DHCP request as documented in Internet draft <draft-ietf-dhc-fqdn-option-12.txt>
4	Configuration settable	1 (TRUE) shall indicate the interface configuration attribute is settable. Some devices, for example a PC or workstation, may not allow the interface configuration to be set via the TCP/IP Interface object
5-31	Reserved	Reserved for future use and shall be set to zero

7.5.3.4 Configuration control

The configuration control attribute is a bitmap used to control network configuration options, as specified in Table 51.

Table 51 – Configuration control bits

Bit	Name	Definition	
0–3	Startup Configuration	Determines how the device shall obtain its initial configuration at start up	0 = The device shall use the network configuration previously stored (for example, in non-volatile memory or via hardware switches, etc) 1 = The device shall obtain its network configuration via BOOTP. 2 = The device shall obtain its network configuration via DHCP upon start-up 3-15 = Reserved for future use
4	DNS Enable	If 1 (TRUE), the device shall resolve host names by querying a DNS server	
5–31	Reserved	Reserved for future use and shall be set to zero	

Devices are not required to support any of the particular values of the Startup Configuration bits, however a device shall support at least one method of obtaining an initial TCP/IP interface configuration. For example, some low-end devices may choose to obtain network interface configuration via BOOTP or DHCP only. Other devices may wish to only allow initial configuration over a serial (or other) port, then store the configuration in non-volatile memory.

The use of BOOTP and/or DHCP may not be appropriate for some devices. DHCP in particular supports dynamically allocated IP addresses, which could result in a device getting a different IP address each time it powers up. This behavior is not appropriate for devices that need to have static IP addresses.

Out of the box, a device may wish to obtain its initial configuration via some method other than BOOTP or DHCP. For example, the device may wish to obtain its initial configuration over an attached serial port, or via another communications port compatible with the corresponding Type 2 Application Layer protocol (assuming the device has another communications port). In this case, the device shall have its startup configuration bits set to 0, and its interface configuration attribute fields to all 0s. The device shall then wait to be configured over another communications port.

Once a device is up and running, when the value of the startup configuration bits is 0, a request to set the interface configuration attribute shall cause the device to store the contents of the interface configuration attribute in non-volatile storage if supported by the device. The startup configuration bits shall not be set to 0 unless the interface configuration attribute has previously been set. Otherwise the device could be rendered unable to communicate on the network.

7.5.3.5 Physical link object

This attribute identifies the object associated with the underlying physical port. There are two components to the attribute : a path size (in UINTs) and a path. The path shall contain a class segment and an instance segment that identify the physical port object. The maximum path size is 6 (assuming a 32 bit logical segment for each of the class and instance). An example path is shown in Table 52.

Table 52 – Example path

Path	Meaning
[20][f6][24][01]	[20] = 8 bit class segment type; [f6] = Ethernet Link object class; [24] = 8 bit instance segment type; [01] = instance 1

The physical link object itself typically maintains link-specific counters as well as any link-specific configuration attributes. The most likely physical link object is the Ethernet Link object (see 7.6).

7.5.3.6 Interface configuration

This attribute contains the configuration parameters required to operate as a TCP/IP node. In order to prevent incomplete or incompatible configuration, the parameters making up the interface configuration attribute cannot be set individually. To modify the Interface configuration attribute, the user should first get the interface configuration attribute, change the desired parameters, then set the attribute.

The TCP/IP Interface object shall apply the new configuration upon completion of the Set service. If the value of the startup configuration bits (configuration control attribute) is 0, the new configuration shall be stored in non-volatile memory. The device shall not reply to the set service until the values are safely stored to non-volatile storage.

If initial configuration is to be obtained via BOOTP or DHCP, the interface configuration attribute components shall be all zeros until the BOOTP or DHCP reply is received. Upon receipt of the BOOTP or DHCP reply, the interface configuration attribute shall show the configuration obtained via BOOTP/DHCP.

Devices are not required to support the Set service. Some implementations, for example those running on a PC or a Workstation, need not support setting the network interface configuration via the TCP/IP Interface object.

Components of the interface configuration attributes are described in Table 53.

Table 53 – Interface configuration components

Name	Meaning
IP address	The device's IP address
Network mask	The device's network mask. The network mask is used when the IP network has been partitioned into subnets. The network mask is used to determine whether an IP address is located on another subnet
Gateway address	The IP address of the device's default gateway. When a destination IP address is on a different subnet, packets are forwarded to the default gateway for routing to the destination subnet
Name server	The IP address of the primary name server. The name server is used to resolve host names. For example, that might be contained in a fieldbusconnection path
Name server 2	The IP address of the secondary name server. The secondary name server is used when the primary name server is not available, or is unable to resolve a host name
Domain name	The default domain name. The default domain name is used when resolving host names that are not fully qualified. For example, if the default domain name is "network.org", and the device needs to resolve a host name of "plc", then the device will attempt to resolve the host name as "plc.network.org"

7.5.3.7 Host name

The Host Name attribute contains the device's host name. The host name attribute is used when the device supports the DHCP-DNS Update capability and has been configured to use DHCP upon start up. The mechanism allows the DHCP client to transmit its host name to the DHCP server. The DHCP server then updates the DNS records on behalf of the client.

NOTE The DHCP-DNS Update mechanism is specified in Internet draft <draft-ietf-dhc-fqdn-option-12.txt>.

The host name attribute does not need to be set for the device to operate normally. The value of the Host Name attribute, if it is configured, shall be used for the value of the FQDN option in the DHCP request. If the Host Name attribute has not been configured then the device shall not include the FQDN option in the DHCP request.

For devices that do not support the DHCP-DNS capability, or are not configured to use DHCP, then the host name may be used for informational purposes.

The set access is optional when the interface configuration attribute is not settable. Some devices (e.g. a PC or workstation), may not allow the interface configuration or host name attributes to be set via the TCP/IP Interface object. If the set is not implemented, an "Attribute not settable" (0x0E) error shall be returned in response to a "Set_Attribute_Single" request.

7.5.3.8 TTL value

TTL value is the value the device shall use for the IP header Time-to-live field when sending CP 2/2 packets via IP multicast. By default, TTL Value shall be 1. The maximum value for TTL is 255. Unicast packets shall use the TTL as configured for the TCP/IP stack, and not the TTL value configured in this attribute.

When set, the TTL Value attribute shall be saved in non-volatile memory, and shall take affect the next time the device starts. Setting the TTL value attribute shall also cause the Mcast pending bit in the Interface status attribute to be set, indicating that there is pending configuration. When a new TTL value is pending, Get_Attribute_Single or Get_Attribute_All requests shall return the pending value. The Mcast pending bit shall be cleared the next time the device starts.

NOTE Users should exercise caution when setting the TTL value greater than 1, to prevent unwanted multicast traffic from propagating through the network. IEC 61158-6-2 includes a discussion on user considerations when using multicast.

7.5.3.9 Mcast config

The Mcast config attribute contains the configuration of the device's IP multicast addresses to be used for CP 2/2 multicast packets. There are three elements to the Mcast config structure: Alloc control, Num mcast, and Mcast start addr.

- Alloc control determines how the device shall allocate IP multicast addresses (e.g., whether by algorithm, whether they are explicitly set,...). Table 54 shows the details for alloc control.

Table 54 – Alloc control values

Value	Definition
0	Multicast addresses shall be generated using the default allocation algorithm specified in IEC 61158-6-2. When this value is specified on a Set_Attribute_Single or Set_Attribute_All, the values of Num mcast and Mcast start addr in the Set_Attribute request shall be 0
1	Multicast addresses shall be allocated according to the values specified in Num mcast and Mcast start addr
2	Reserved

- Num mcast is the number of IP multicast addresses allocated. The maximum number of multicast addresses is device specific, but shall not exceed the number of CP 2/2 multicast connections supported by the device.
- Mcast start addr is the starting multicast address from which Num mcast addresses are allocated.

When set, the Mcast config attribute shall be saved in non-volatile memory, and shall take affect the next time the device starts. Setting the Mcast config attribute shall also cause the

Mcast pending bit in the Interface status attribute to be set, indicating that there is pending configuration. When a new Mcast Config value is pending, Get_Attribute_Single or Get_Attribute_All requests shall return the pending value. The Mcast pending bit shall be cleared the next time the device starts.

When the multicast addresses are generated using the default algorithm, Num mcast and Mcast start addr shall report the values generated by the algorithm.

7.5.4 Common services

7.5.4.1 General

The TCP/IP Interface object shall support the common services as specified in Table 55.

Table 55 – TCP/IP Interface object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Optional	Get_Attribute_All	Returns a predefined listing of this objects attributes (see the Get_Attribute_All response definition in 7.5.4.2)
0x02	N/A	Optional	Set_Attribute_All	Modifies all settable attributes
0x0E	Conditional ^a	Required	Get_Attribute_Single	Returns the contents of the specified attribute
0x10	N/A	Required	Set_Attribute_Single	Modifies a single attribute

^a The Get_Attribute_Single service shall be implemented for Class if any class attributes are implemented.

7.5.4.2 Get_Attribute_All response

For class attributes, since there is only one class attribute, class attribute #1 shall be returned.

For instance attributes, attributes shall be returned in numerical order up to the last implemented attribute. The Get_Attribute_All reply shall be as specified in Table 56.

Table 56 – Get_Attribute_All reply format

Attribute ID	Size (octets)	Contents
1	4	Status
2	4	Configuration capability
3	4	Configuration control
4	2	Physical link object, path size
	Variable, 12 octets max	Physical link object, path (if path size is non-zero)
5	4	IP address
	4	Network mask
	4	Gateway address
	4	Name server
	4	Secondary name server
	2	Domain name length
	Variable, equal to domain name length	Domain name
	1	Pad octet only if Domain Name Length is odd
6	2	Host name length
	variable, equal to host name length	Host name
	1	Pad octet only if Host Name Length is odd
7	6	See IEC 61784-3-2. Not present if attribute 7 is not implemented. Shall be 0 when additional attributes greater than attribute ID 7 are included
8	1	TTL value. Not present if attribute 8 is not implemented. Shall be 0 when additional attributes greater than attribute ID 8 are included
9	8	Mcast config. Not present if attribute 9 is not implemented. Shall be 0 when additional attributes greater than attribute ID 9 are included

The lengths of the physical link object path, domain name, and host name are not known before issuing the Get_Attribute_All service request. Implementers shall be prepared to accept a response containing the maximum sizes of the physical link object path (6 UINTs), the domain name (48 USINTs), and host name (64 USINTs).

7.5.4.3 Set_Attribute_All request

The instance Set_Attribute_All request shall contain the configuration control attribute, followed by the interface configuration attribute.

7.5.5 Behavior

The behavior of the TCP/IP Interface object shall be as illustrated in the State Transition Diagram shown in Figure 21.

Note that the act of obtaining an initial executable image via BOOTP/TFTP shall not be considered within the scope of the TCP/IP Interface object behavior. Devices are free to implement such behavior, however it shall be considered to have occurred in the “Non-existent” state.

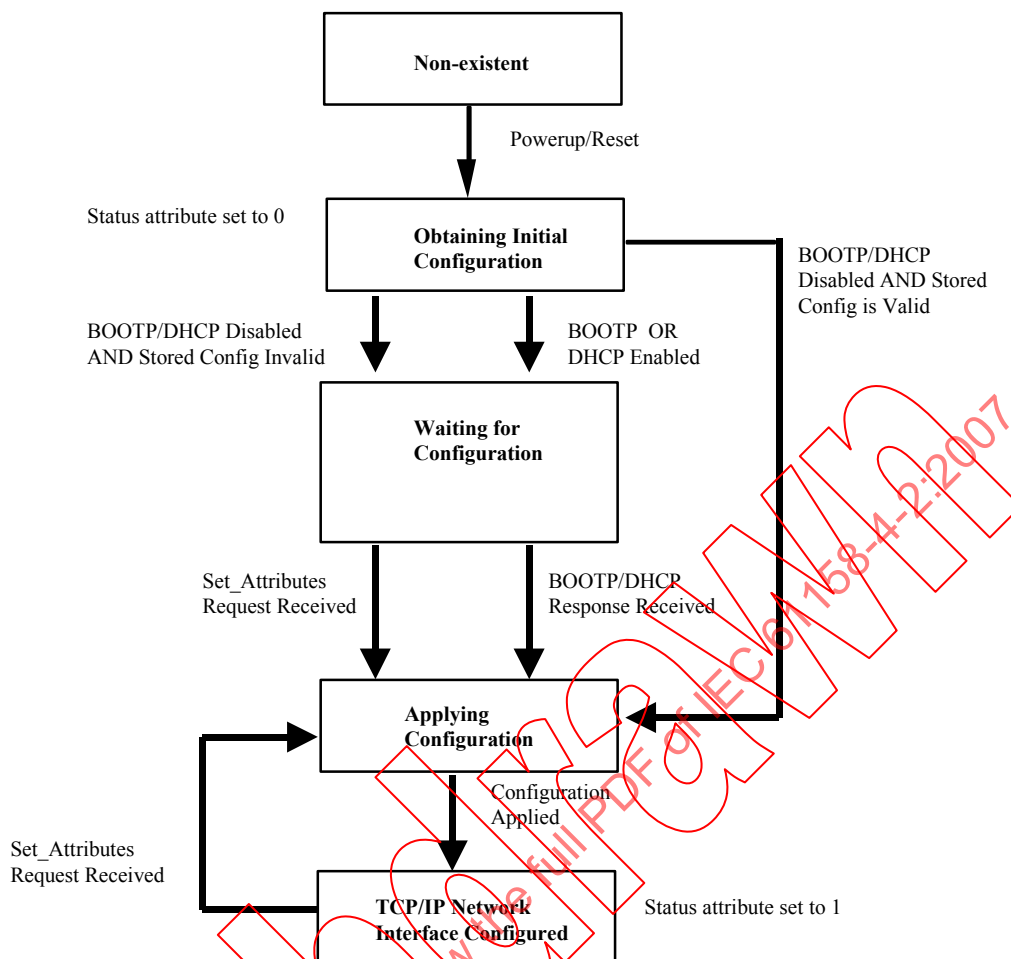


Figure 21 – State transition diagram for TCP/IP Interface object

7.6 Ethernet link object

7.6.1 Overview

The Ethernet Link object maintains link-specific counters and status information for a physical Ethernet ISO/IEC 8802-3 port. Each device shall support exactly one instance of the Ethernet Link object for each Ethernet port on the module. A request to access instance 1 of the Ethernet Link object shall always refer to the instance associated with the port over which the request was received.

7.6.2 Revision history

The revision history of the Ethernet Link object is described in Table 57.

Table 57 – Ethernet link object revision history

Class revision	Description of changes
01	Initial Release
02	Release for IEC 61158.

7.6.3 Class attributes

The Ethernet Link object shall support the class attributes as specified in Table 58.

Table 58 – Ethernet link object class attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x01	Required	Get	revision	UINT	revision of this object	current value for this attribute = 2
0x02 to 0x07	These class attributes are optional and are described in IEC 61158-5-2					

An error reading the class revision attribute implies this is a revision 1 only implementation.

7.6.4 Instance attributes

7.6.4.1 General

The Ethernet Link object shall support the instance attributes as specified in Table 59.

Table 59 – Ethernet link object instance attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x01	Required	Get	Interface speed	UDINT	Interface speed currently in use	Speed in Mbit/s (e.g. 0, 10, 100, 1 000). See 7.6.4.2
0x02	Required	Get	Interface flags	DWORD	Interface status flags	Bit map of interface flags. See 7.6.4.3
0x03	Required	Get	Physical address	USINT[6]	MAC layer address	See 7.6.4.4
0x04	Conditional ^a	Get	Interface counters	STRUCT of 44 octets		See 7.6.4.5
			In Octets	UDINT	Octets received on the interface	
			In Ucast Packets	UDINT	Unicast packets received on the interface	
			In NUcast Packets	UDINT	Non-unicast packets received on the interface	
			In Discards	UDINT	Inbound packets received on the interface but discarded	
			In Errors	UDINT	Inbound packets that contain errors (does not include In Discards)	
			In Unknown Protos	UDINT	Inbound packets with unknown protocol	
			Out Octets	UDINT	Octets sent on the interface	
			Out Ucast Packets	UDINT	Unicast packets sent on the interface	

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
			Out NUCast Packets	UDINT	Non-unicast packets sent on the interface	
			Out Discards	UDINT	Outbound packets discarded	
			Out Errors	UDINT	Outbound packets that contain errors	
0x05	Optional	Get	Media counters	STRUCT of 48 octets	Media-specific counters	See 7.6.4.6
			Alignment errors	UDINT	frames received that are not an integral number of octets in length	
			FCS errors	UDINT	DLPDUs received that do not pass the FCS check	
			Single collisions	UDINT	Successfully transmitted DLPDUs which experienced exactly one collision	
			Multiple collisions	UDINT	Successfully transmitted DLPDUs which experienced more than one collision	
			SQE test errors	UDINT	Number of times SQE test error message is generated	
			Deferred transmissions	UDINT	DLPDUs for which first transmission attempt is delayed because the medium is busy	
			Late collisions	UDINT	Number of times a collision is detected later than 512 bit-times into the transmission of a packet	
			Excessive collisions	UDINT	DLPDUs for which transmission fails due to excessive collisions	
			MAC transmit errors	UDINT	DLPDUs for which transmission fails due to an internal MAC sublayer transmit error	
			Carrier sense errors	UDINT	Times that the carrier sense condition was lost or never asserted when attempting to transmit a frame	
			Frame too long	UDINT	DLPDUs received that exceed the maximum permitted DLPDU size	
			MAC receive errors	UDINT	DLPDUs for which reception on an interface fails due to an internal MAC sublayer receive error	

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x06	Optional	Get/Set	Interface control	STRUCT of 4 octets	Configuration for physical interface	See 7.6.4.7
			Control bits	WORD	Interface control bits	
			Forced interface speed	UINT	Speed at which the interface shall be forced to operate	Speed in Mbit/s (e.g. 10, 100, 1 000)
a The interface counters attribute is required if the media counters attribute is implemented.						

7.6.4.2 Interface speed

The interface speed attribute shall indicate the speed at which the interface is currently running (e.g. 10 Mbit/s, 100 Mbit/s, 1 Gbit/s, or other speeds). A value of 0 shall be used to indicate that the speed of the interface is indeterminate.

The scale of the attribute is in Mbit/s, so if the interface is running at 100 Mbit/s then the value of the interface speed attribute shall be 100. The interface speed is intended to represent the available link transmit time; the attribute shall not be doubled if the interface is running in full-duplex mode.

7.6.4.3 Interface flags

The interface flags attribute contains status and configuration information about the physical interface as specified in Table 60.

Table 60 – Interface flags bits

Bit	Name	Definition
0	Link status	Indicates whether or not the port is connected to an active network : - 0 indicates an inactive link - 1 indicates an active link The determination of link status is implementation specific. In some cases devices can tell whether the link is active via hardware/driver support. In other cases, the device may only be able to tell whether the link is active by the presence of incoming packets
1	Half/Full duplex	Indicates the duplex mode currently in use: - 0 indicates the port is running half duplex - 1 indicates full duplex If the Link status flag is 0, then the value of the Half/Full duplex flag is indeterminate.
2-4	Negotiation status	Indicates the status of link auto-negotiation: 0 = Auto-negotiation in progress 1 = Auto-negotiation and speed detection failed. Using default values for speed and duplex. Default values are product-dependent; recommended defaults are 10Mbit/s and half duplex 2 = Auto negotiation failed but detected speed. Duplex was defaulted. Default value is product-dependent; recommended default is half duplex 3 = Successfully negotiated speed and duplex 4 = Auto-negotiation not attempted. Forced speed and duplex
5	Manual setting requires reset	0 indicates the interface can activate changes to link parameters (auto-negotiate, duplex mode, interface speed) automatically 1 indicates the device requires a Reset service be issued to its Identity object in order for the changes to take effect

Bit	Name	Definition
6	Local hardware fault	0 indicates the interface detects no local hardware fault 1 indicates a local hardware fault is detected The meaning of this is product-specific. Examples are an AUI/MII interface detects no transceiver attached or a radio modem detects no antennae attached. In contrast to the soft, possible self-correcting nature of the Link Status being inactive, this is assumed a hard-fault requiring user intervention
7-31	Reserved	Reserved and shall be set to zero

7.6.4.4 Physical address

The physical address attribute contains the interface's MAC layer address. The physical address is an array of octets. The recommended display format is "XX-XX-XX-XX-XX-XX", starting with the first octet. The physical address is not a settable attribute: the Ethernet address shall be assigned by the manufacturer, and shall be unique per ISO/IEC 8802-3 requirements.

7.6.4.5 Interface counters

The interface counters attribute contains counters relevant to the receipt of packets on the interface. These counters shall be as defined in RFC 1213 "MIB-II Management Information Base". The interface counters is a conditional attribute: it shall be implemented if the media counters attribute is implemented.

7.6.4.6 Media counters

The media counters attribute contains counters specific to Ethernet media. These counters shall be as defined by RFC 1643, "Definitions of Managed Objects for Ethernet-Like Interface Types". If this attribute is implemented the interface counters attribute shall also be implemented.

7.6.4.7 Interface control

7.6.4.7.1 Overview

The interface control attribute is a structure consisting of control bits and forced interface speed and shall be as specified in the following subclauses.

7.6.4.7.2 Control bits

The control bits attribute is specified in Table 61.

Table 61 – Control bits

Bit	Name	Definition
0	Auto-negotiate	0 indicates ISO/IEC 8802-3 link auto-negotiation is disabled 1 indicates auto-negotiation is enabled If auto-negotiation is disabled, then the device shall use the settings indicated by the forced duplex mode and forced interface speed bits
1	Forced duplex mode	If the Auto-negotiate bit is 0, the forced duplex mode bit indicates whether the interface shall operate in full or half duplex mode. 0 indicates the interface duplex should be half duplex 1 indicates the interface duplex should be full duplex Interfaces not supporting the requested duplex shall return an error code 0x09 (Invalid Attribute Value). If auto-negotiation is enabled, attempting to set the forced duplex mode bits shall result in an error code 0x0C (Object State Conflict)
2-15	Reserved	Shall be set to zero

7.6.4.7.3 Forced interface speed

If the auto-negotiate bit is 0, the forced interface speed bits indicate the speed at which the interface shall operate. Speed is specified in Mbit/s (e.g. for 10 Mbit/s Ethernet, the interface speed shall be 10). Interfaces not supporting the requested speed should return an error code 0x09 (Invalid Attribute Value).

If auto-negotiation is enabled, attempting to set the Forced Interface Speed shall result in an error code 0x0C (Object State Conflict).

7.6.5 Common services**7.6.5.1 General**

The Ethernet Link object shall support the common services as specified in Table 62.

Table 62 – Ethernet Link object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Optional	Get_Attribute_All	Returns a predefined listing of this objects attributes (see the Get_Attribute_All response definition in 7.6.5.2)
0x0E	Conditional	Required	Get_Attribute_Single	Returns the contents of the specified attribute
0x10	N/A	Conditional	Set_Attribute_Single	Modifies a single attribute

The Get_Attribute_Single shall be implemented for the class attribute if the class attribute is implemented.

The Set_Attribute_Single service shall be implemented if the interface control attribute is implemented.

7.6.5.2 Get_Attribute_All response

For class attributes, since there is only one possible attribute, the Get_Attribute_All response is the same as the Get_Attribute_Single response. If no class attributes are implemented, then no data is returned in the data portion of the reply.

For instance attributes, attributes shall be returned in numerical order, up to the last implemented attribute. All zeros shall be returned for the interface counters and media counters attributes if they are not implemented but the interface control attribute is implemented.

7.6.6 Class specific services

7.6.6.1 General

The Ethernet Link object shall support the class specific services as specified in Table 63.

Table 63 – Ethernet Link object class specific services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x4C	N/A	Conditional	Get_and_Clear	Gets then clears the specified attribute (interface counters or media counters)

The Get_and_Clear service shall only be implemented if the interface counters and media counters attributes are implemented.

7.6.6.2 Get_and_Clear service

The Get_and_Clear service is a class specific service. It is only supported for the interface counters and media counters attributes. The Get_and_Clear response shall be the same as the Get_Attribute_Single response for the specified attribute. After the response is built, the value of the attribute shall be set to zero.

7.7 DeviceNet object

7.7.1 Overview

The DeviceNet object shall provide a consistent Station Management interface to the Physical and Data Link Layers. This object shall make diagnostic information from these layers available to client applications. Each node shall support one DeviceNet object per link.

NOTE A device may contain more than one node.

7.7.2 Revision history

The revision history of the DeviceNet object is described in Table 64.

Table 64 – DeviceNet object revision history

Class revision	Description of changes
01	Initial Release
02	Release for IEC 61158 series

7.7.3 Class attributes

The DeviceNet object shall support the class attributes as specified in Table 65.

Table 65 – DeviceNet object class attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x01	Required	Get	Revision	UINT	revision of this object	2 (revision 2)

7.7.4 Instance attributes

7.7.4.1 General

The DeviceNet object shall support the instance attributes as specified in Table 66.

Table 66 – DeviceNet object instance attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x01	Optional	Get/Set	MAC ID	USINT	node address	see 7.7.4.2
0x02	Optional	Get/Set	Bit Rate	USINT	bit rate	see 7.7.4.3
0x03	Optional	Get/Set	BOI	BOOL	bus-off interrupt	see 7.7.4.4
0x04	Optional	Get/Set	Bus-Off Counter	USINT	number of times CAN went to the bus-off state	see 7.7.4.5
0x05	Optional	Get	Allocation Information	STRUCT of 2 octets		see 7.7.4.6
			Allocation choice	SWORD		see 7.7.4.6.2
			Master's MAC ID	USINT	MAC ID of master (from allocate)	see 7.7.4.6.3
0x06	Conditional ^a	Get	MAC ID switch changed	BOOL	The node address switch(es) have changed since last power-up/reset	0 = no Change 1 = change since last reset or power-up
0x07	Conditional ^b	Get	Bit rate switch changed	BOOL	The bit rate switch(es) have changed since last power-up/reset	0 = no Change 1 = change since last reset or power-up
0x08	Conditional ^a	Get	MAC ID switch value	USINT	Actual value of node address switch(es)	0 to 99
0x09	Conditional ^b	Get	Bit rate switch value	USINT	Actual value of bit rate switch(es)	0 to 9
0x0A	Optional ^c	Get/Set	Quick Connect	BOOL	Enable/Disable of Quick Connect feature	0 = disable (default) 1 = enable
0x0B	Conditional ^d		Safety Network Number	SWORD[6]		See IEC 61784-3-2

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x0C	Optional	Get	Diagnostic counters	STRUCT of 34 octets	List of ISO 11898 diagnostic counters	See 7.7.4.7
			Diagnostic counters descriptor	WORD	Indicates which diagnostic counters are supported	
			Arbitration loss count	UINT	See ISO 11898	Range = 0 – 65 535 Default = 0
			Overload count	UINT	See ISO 11898	
			Bit error count	UINT	See ISO 11898	
			Stuff error count	UINT	See ISO 11898	
			Ack error count	UINT	See ISO 11898	
			Form error count	UINT	See ISO 11898	
			CRC error count	UINT	See ISO 11898	
			Rx message loss count	UINT	ISO 11898 subsystem has detected a lost receive message	Range = 0 – 256
			Warning error count	UINT	See ISO 11898	
			Rx error counter	UINT	Receive error counter (see ISO 11898)	
			Tx error counter	UINT	Transmit error counter (see ISO 11898)	Range = 0 – 256
				UINT[5]	Reserved	Default = 0
^a These attributes are required if the MAC ID switch can be set to a different value than the device is currently on-line at. ^b These attributes are required if the bit rate switch can be set to a different value than the device is currently on-line at. ^c This attribute shall be stored in non-volatile memory. ^d This attribute is required for CPF 2 safety devices. Non-safety devices shall not implement this attribute.						

7.7.4.2 MAC ID

This attribute contains the MAC ID of this device. The range of values is 0 to 63 decimal.

A device that uses a switch to set the MAC ID shall return an Error Response, with General Error Code 0x0E (Attribute not settable), in response to a Set_Attribute_Single request when the switch setting is 0 to 63 and the MAC ID switch is enabled by other settings. When the MAC ID switch setting is greater than 63 (disabled) or the MAC ID switch is disabled by other settings, the device may support the setting of the MAC ID attribute by the Set_Attribute_Single service request. The device shall provide visual indication (e.g. via physical switch, LED) that the MAC ID switch has been overridden. A minor recoverable fault shall be declared when the MAC ID switch is enabled and indicates a valid MAC ID, but does not match the current on-line address of the device. During power up or reset, a device should go to the Communications Faulted state if it has a MAC ID switch set to an invalid setting and does not support the setting of the MAC ID attribute.

The MAC ID attribute is considered non-volatile in that once configured the attribute shall be remembered after a power cycle or device reset. The “out-of-box” configuration shall default to a MAC ID value of 63. The following MAC ID determination scenarii apply only to devices that use non-volatile memory to store the MAC ID of the device.

1. When the MAC ID switch is valid and enabled, the switch value is loaded into non-volatile memory when the device powers up or is reset, and prior to attempt to go on-line. The MAC ID switch value is not loaded into non-volatile memory at any other time.
2. Devices shall attempt to go on-line with the MAC ID value stored in non-volatile memory or, if in the “out-of-box” configuration, with a MAC ID value of 63.
3. When a Set_Attribute_Single request with a valid MAC ID is received by a device with the MAC ID switch disabled, the non-volatile MAC ID shall be loaded with the new MAC ID.
4. When a Set_Attribute_Single request with a valid MAC ID is received by a device with the MAC ID switch enabled, the non-volatile MAC ID shall not change and an Error Response with General Error Code 0x0E (Attribute not settable) shall be returned.

The modification of the MAC ID requires a device to delete all Connection objects and re-execute the Network Access State Machine defined in IEC 62026-3.

7.7.4.3 Bit Rate

The Bit Rate attribute indicates the selected bit rate. Values are specified in Table 67.

Table 67 – Bit rate attribute values

Value	Meaning
00	125 kbit/s
01	250 kbit/s
02	500 kbit/s

A device that uses a switch to set the bit rate shall return an Error Response, with General Error Code 0x0E (Attribute not settable), in response to a Set_Attribute_Single request when the bit rate switch is set to a valid value and not disabled by other settings. When the bit rate switch is not set to a valid value (disabled) and/or the bit rate switch is disabled by other settings, the device may support the setting of the Bit Rate attribute by the Set_Attribute_Single service request. The device shall provide visual indication (via physical switch, LED, etc.) that the bit rate switch has been overridden. A minor recoverable fault shall be declared when the bit rate switch is enabled and indicates a valid bit rate, but does not match the current on-line bit rate of the device. During power up or reset, a device should go to the Communication Fault state if it has a bit rate switch set to an invalid setting and does not support the setting of the Bit Rate attribute.

The Bit Rate attribute is considered non-volatile in that once configured the attribute shall be remembered after a power cycle or device reset. The “out-of-box” configuration shall default such that the device is able to go online at 125 kbit/s. The following bit rate determination scenarii apply only to devices that use non-volatile memory to store the bit rate of the device.

1. When the bit rate switch is valid and enabled, the switch value is loaded into non-volatile memory when the device powers up or is reset, and prior to attempting to go online. The bit rate switch value is not loaded into non-volatile memory at any other time.
2. Devices shall attempt to go on-line with the bit rate value stored in non-volatile memory or, if in the “out-of-box” configuration, able to go online at 125 kbit/s.
3. When a Set_Attribute_Single request with a valid bit rate is received by a device with the bit rate switch disabled, the non-volatile Bit Rate shall be loaded with the new bit rate.
4. When a Set_Attribute_Single request with a valid bit rate is received by a device with the bit rate switch enabled, the non-volatile Bit Rate shall not change, and an Error Response with General Error Code 0x0E (Attribute not settable) shall be returned.

The modification of the Bit Rate will not take effect until the device is either physically reset (such as cycle of power or a reset switch) or reset by sending the Reset Service to the Identity object. During this time the Bit Rate attribute value will not match the actual network bit rate.

7.7.4.4 BOI (Bus-off interrupt)

The BOI attribute consists of one bit that defines how a CAN device processes the bus-off interrupt. The BOI attribute is bit position 0 within an octet for the Get_Attribute_Single/Set_Attribute_Single services. The rest of the bits in the octet shall be zeros.

Table 68 – BOI attribute values

Value	Meaning
00	Hold the CAN chip in its bus-off (reset) state upon detection of a bus-off indication
01	If possible, fully reset the CAN chip and continue communicating upon detection of a bus-off indication

When the BOI attribute is FALSE (set to zero) and an ISO 11898 CAN chip bus-off event is detected, the following steps are taken:

- the CAN chip is held in its reset/bus-off state,
- the device enters the Communications Faulted state (see IEC 62026-3).

When the BOI attribute is TRUE (set to one) and a CAN chip bus-off event is detected, it may be possible to return the CAN chip to its normal operating mode and continue communicating based on the Network Access State Machine presented in IEC 62026-3.

If the BOI attribute is set to one the device shall insure that it does not perpetually reset and continue to produce corrupted packets on the bus. Failing to do so will allow the device to disrupt all communications on the bus.

Connections are not necessarily effected when a bus-off event is detected and the device is able to continue communicating. Previously established connections can remain existing or they can be deleted (soft reset). Either way, the Duplicate MAC ID detection algorithm shall be performed again per the Network Access State Machine.

As indicated in Table 66, support of the BOI attribute is optional. If it is not supported, a device shall implement the behavior indicated by attribute value zero (0) in Table 68.

7.7.4.5 Bus-off counter

The Bus-off Counter counts the number of times the CAN chip went to the bus-off state (counts number of bus-off interrupts). The counter has values of 0 to 255 decimal.

The Bus-off Counter is initialized to zero at power-up or device initialization.

The Bus-off Counter stops counting when it reaches maximum count. The counter does not roll over. The Counter stays at maximum count until a Set_Attribute_Single is performed.

The DeviceNet object resets the Bus-off Counter to zero (0) whenever it receives a Set_Attribute_Single request specifying the Bus-Off Counter attribute. The Set_Attribute_Single data is not used and can be any value. The transmission of a Set_Attribute_Single request to the Bus-off Counter is all that's required to reset the counter.

7.7.4.6 Allocation information

7.7.4.6.1 Overview

The Allocation Information attribute is pertinent to the Predefined Master/Slave Connection Set. Its support is required if the Predefined Master/Slave Connection Set is supported. It indicates whether or not the Predefined Master/Slave Connection Set defined in IEC 62026-3 has been allocated. If it has been allocated, then, this attribute indicates the device that has performed the allocation and the Connection(s) that are currently allocated.

This attribute is modified when a successful response associated with an Allocate_Master/Slave_Connection_Set service (defined later in this section) is generated. This attribute is not modifiable by the Set_Attribute_Single Service. An Error Response whose General Error Code Field is set to 0x0E (Attribute not settable) is returned if a Set_Attribute_Single request specifies this attribute.

The Allocation Information attribute consists of the Allocation choice and the Master's MAC ID.

7.7.4.6.2 Allocation choice

The Allocation choice indicates which of the Predefined Master/Slave Connections are active (in the Configuring, or Established state). Its format is specified in IEC 62026-3.

The Allocation choice is initialized to 00 at device power-up or reset.

7.7.4.6.3 Master's MAC ID

The Master's MAC ID contains the MAC ID of the device that has allocated the Predefined Master/Slave Connection Set via the Allocate_Master/Slave_Connection_Set service. This contains the Allocator's MAC ID field copied from the Allocate_Master/Slave_Connection_Set request.

The range of values is 0 to 63 and 255 decimal. A value in the range 0–63 indicates that the Predefined Master/Slave Connection Set is currently allocated and denotes the MAC ID of the device that performed the allocation. The value 255 means the Predefined Master/Slave Connection set has not been allocated.

The Master's MAC ID attribute is initialized to 255 (0xFF) at device power-up/reset.

7.7.4.7 Diagnostic counters

This attribute reports the counts of various types of network errors. The first field, diagnostic counters descriptor, identifies which counters are supported by the device. Each bit indicates if a designated counter is supported. If set, the counter is supported. Table 69 specifies the diagnostic counter bit assignment.

Table 69 – Diagnostic counters bit description

Bit	Counter	Clearable
0	Arbitration loss	Yes
1	Overload error	Yes
2	Bit error	Yes
3	Stuff error	Yes
4	Ack error	Yes
5	Form error	Yes
6	CRC error	Yes

Bit	Counter	Clearable
7	Rx message loss	Yes
8	Warning error	Yes
9	Rx error	No
10	Tx error	No
11 – 15	Reserved (shall be 0)	N/A

The remaining fields provide counter values for each supported counter. The counters are advanced by one for each occurrence of the error, except for the Rx and Tx error counters (fields 10 and 11) which are incremented and decremented by the CAN peripheral based on the ISO 11898 standard, and do not roll over.

All counters are cleared to zero when the device enters the Sending Duplicate MAC ID Check Request Message (see the Link Access State Transition Diagram in IEC 62026-3).

7.7.5 Common services

7.7.5.1 General

The DeviceNet object shall support the common services as specified in Table 70.

Table 70 – DeviceNet object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x05	N/A	Conditional ^a	Reset	Used to reset this instance of the DeviceNet object
0x10	N/A	Optional	Set_Attribute_Single	Set network or node specific configuration information
0x0E	Optional	Optional	Get_Attribute_Single	Get network or node specific configuration information

^a This service is required when this DeviceNet object instance is addressable through some other CPF 2 access mechanism other than the subnet interface controlled by this DeviceNet object instance.

7.7.5.2 Reset

When the DeviceNet object receives a Reset request, it

- determines if it can provide the type of reset requested;
- responds to the request;
- performs the type of reset requested.

The Reset common service has the object-specific parameter specified in Table 71.

Table 71 – Reset service parameter

Name	Type	Description of request parameters	Semantics of values
Type	USINT	Type of reset	See Table 72

The parameter Type for the Reset common service has the enumeration specifications shown in Table 72.

Table 72 – Reset service parameter values

Value	Type of reset
0	Emulate as closely as possible cycling power on the network link that the DeviceNet object instance represents. This value is the default if this parameter is omitted
1	Return as closely as possible to the out-of-box configuration, then emulate cycling power on the network link as closely as possible
2	Excepting the MAC ID and bit rate, return as closely as possible to the out-of-box configuration, then emulate cycling power on the network link as closely as possible. The MAC ID and bit rate is not changed by this reset
3 – 99	Reserved
100– 199	Vendor specific reset behavior
200 – 255	Reserved

7.7.6 Class specific services

7.7.6.1 General

The DeviceNet object shall support the class specific services as specified in Table 73.

Table 73 – DeviceNet object class specific services

Service code	Need in implementation	Service name	Description of service
0x4B	Optional	Allocate_Master/Slave_Connection_Set	Requests the use of the Predefined Master/Slave Connection Set
0x4C	Conditional ^a	Release_Master/Slave_Connection_Set	Indicates that the specified connections within the Predefined Master/Slave Connection Set are no longer desired. These connections shall be released (deleted)
0x4D	Conditional ^b	Clear_Diagnostics	Clears the diagnostic counters that are reported in attribute 12
^a The Release_Master/Slave_Connection_Set service is required if the Allocate_Master/Slave_Connection_Set service is implemented in the device. ^b This service shall be supported if any clearable counters are implemented in the Diagnostic counters attribute.			

7.7.6.2 Allocate_Master/Slave_Connection_Set, Release_Master/Slave_Connection_Set

These services are used to allocate and deallocate the Predefined Master/Slave Connection Set described in IEC 62026-3.

A device that behaves as the client across the Predefined Master/Slave Connection Set is referred to as a master. A device that behaves as the server across the Predefined Master/Slave Connection Set is referred to as a slave. Within the bounds of the service descriptions to follow, a master and/or slave is viewed as a functional unit within a communicating device.

A device that wants to function as another's master shall first allocate the Predefined Master/Slave Connection Set within the slave. Only one master can have the Predefined Master/Slave Connection Set allocated at any given time. The entire Connection Set is

allocated and the master uses a select subset of the connections from the set (i.e. Bit Strobe only, or Poll only). When a master wants to “give-up” its slave it releases all connections, causing the slave to “deallocate” the Predefined Master/Slave Connection Set.

7.7.6.3 Clear_Diagnostics

This service clears (to zero) the clearable counters within attribute 12 (diagnostic counters). The counters that are clearable are indicated in Table 69.

7.8 Connection configuration object (CCO)

7.8.1 Overview

This object defines an interface used to create, configure and control connections in a device. This specification does not define or constrain the operation of the device's connection management state machine. The class attributes used by this object are shown in Table 75.

7.8.2 Revision history

The revision history of the Connection Configuration object is described in Table 74.

Table 74 – Connection configuration object revision history

Class revision	Description of changes
01	Initial definition
02	Intermediate definition
03	Release for IEC 61158

7.8.3 Class attributes

7.8.3.1 General

The Connection Configuration object shall support the class attributes as specified in Table 75.

Table 75 – Connection configuration object class attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
0x01	Required	Get	NV	Revision	UINT	Revision of this object	Third revision, value = 3 (see 7.8.2)
0x02	Required	Get	NV	Max instance	UDINT	Maximum instance number	Value determined by node specifics
0x03	Required	Get	NV	Num instance	UDINT	Number of connections currently instantiated	
0x04	This class attribute is optional and is described in IEC 61158-5-2						
0x05	Conditional ^e	Get	NV	Optional service list	STRUCT of variable size	List of optional services utilized in an object class implementation	A list of service codes specifying the optional services implemented in the device for this class
				Number services	UINT	Number of services in the optional service list	The number of service codes in the list
				Optional services	ARRAY of UINT	List of optional service codes	The optional service codes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
0x06 0x07	These class attributes are optional and are described in IEC 61158-5-2						
0x08	Required	Get	NV	Format number	UDINT	See 7.8.3.2	
0x09	Required	Get/Set	NV	Edit signature	UDINT	See 7.8.3.3	
0x0A – 0x20	Reserved						
0x21	Optional	Get/ Set ^{a,b}	NV	Scanner Mode	BOOL	The originator to target packets for connections originated by this device shall reflect the state of this attribute	0 = idle mode 1 = run mode
0x22	Optional	Get	V	Scanner capabilities ^c	WORD	Bit 0, set to Scanner Mode (attribute 0x21) supported ^d Bit 1, set to Scanner Mode (attribute 0x21) currently supported Bit 2, Instances may currently be created in Run mode Bit 3, Instances may currently be changed in Run mode Bit 4, Instances may currently be deleted in Run mode Bit 5, Instance may currently be created in Idle mode Bit 6, Instances may currently be changed in Idle mode Bit 7, Instances may currently be deleted in Idle mode Bit 8, Open_Connection/Close_Connection services supported Bit 9, Stop_Connection service supported Bit 10, Get_Member service to read image tables supported Bits 11-15 – reserved and shall be zero	Bits 0 – 10 = 0, No = 1, Yes

^a A set to attribute 0x21 shall return a device state conflict (0x10) error if changing the Scanner Mode is not permitted when the set attribute command is received.

^b The value of attribute 0x21 may be different than the value last set to attribute 0x21 if another mechanism (like a key switch) changed the scanner's mode.

^c If the device in which this object is implemented has some sort of mechanism for changing connections, the state of these bits shall be changed to reflect the present state of the device

^d Bit 0 indicates if the Scanner Mode attribute has the ability to ever be set using a set attribute service.

^e Required if object supports any optional services. Optional if object does not support any optional services.

7.8.3.2 Format number

This number determines the format of instance attribute 0x09. This format value shall be specified in the format_number field of each instance attribute 0x09. Meaning of the format values is specified in Table 76.

Table 76 – Format number values

Value	Meaning
0	Single O->T/T->O tables, 16-bit words, 0-based offsets
1	Multiple O->T/T->O tables, 16-bit words, 0-based offsets
2 – 99	Reserved
100 – 199	Vendor Specific
200 – ...	All other values are reserved

7.8.3.3 Edit signature

Created and used by configuration software to detect modification to the instance attribute values. For CP 2/1 this value is initially 0 and is incremented each time a change complete operation is performed. For all other CPF 2 networks this value is initially 0 and is set to a 32 bit CRC each time a change complete operation is performed (the polynomial is $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+x^0$), the CRC seed value shall be 0. The CRC is calculated on the set all attribute data stream for each connection instance (lowest to highest) plus class attribute 0x01, class attribute 0x02, class attribute 0x03 and class attribute 0x08.

7.8.4 Instance attributes

7.8.4.1 General

The Connection Configuration object shall support the instance attributes as specified in Table 77.

Table 77 – Connection configuration object instance attributes

Attribute ID	Need in implem	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
0x01	Required	Get	V	Connection status	STRUCT of 4 octets	Connection status	See 7.8.4.2
				Gen_status	USINT	General status	
				Reserved	USINT	Reserved	Shall be zero
				Ext_status	UINT	Extended status	
0x02	Required	Get/Set	NV	Connection flags	WORD	Connection flags	See 7.8.4.3
0x03	Required	Get/Set	NV	Target device ID	STRUCT of 8 octets		See 7.8.4.4
				Vendor_id	UINT	Vendor ID	
				Product_type	UINT	Device type	
				Product_code	UINT	Product code	
				Major_rev	USINT	Major revision	
				Minor_rev	USINT	Minor revision	
0x04	Conditional	Get/Set	NV	CS_data_indx number	UDINT		See 7.8.4.5
0x05	Required	Get/Set	NV	Net connection parameters	STRUCT of 14 octets		See 7.8.4.6
				Conn_timeout	USINT	Connection Timeout Multiplier	
				Xport_class_and_trig	SWORD	Transport Class and Trigger	
				Rpi_OT	UDINT	Originator to Target Requested Packet Interval	
				Net_OT	UINT	Originator to Target network connection parameters	
				Rpi_TO	UDINT	Target to Originator Requested Packet Interval	
				Net_TO	UINT	Target to Originator network connection parameters	
0x06	Required	Get/Set	NV	Connection path	STRUCT of variable size		See 7.8.4.7
				Open_path_size	USINT	Open connection path size	
				Reserved	USINT	Reserved	
				Open connection path	Padded EPATH	Connection path	

Attribute ID	Need in implem	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
0x07	Required	Get	NV	Config #1 data	STRUCT of variable size		See 7.8.4.8
				Config_data_size	UINT	Length of config_data in octets	
				Config_data	ARRAY of octets	Config #1 data	
0x08	Required	Get/Set	NV	Connection Name	STRUCT of variable size		See 7.8.4.9
				name_size	USINT	Number of characters in the connection name.	
				Reserved	USINT	Reserved	Shall be zero
				connection_name	STRING2	User assigned connection name encoded in UNICODE	
0x09	Required	Get/Set	NV	I/O mapping	STRUCT of variable size		See 7.8.4.10
				format_number	UINT	This number determines the format of this attribute	The format value shall match the value of class attribute 0x08
				mapping_data_size	UINT	Size, in octets, of the mapping_data field that follows	
				mapping_data	ARRAY of octets	I/O mapping information associated with this instance	
0x0A	Required	Get/Set	NV	Config #2 data	STRUCT of variable size		See 7.8.4.11
				Config_data_size	UINT	Length of config_data in octets	
				Config_data	ARRAY of octets	Config #2 data	
0x0B	Required	Get/Set	NV	Proxy device ID	STRUCT of 8 octets		See 7.8.4.12
				Vendor_id	UINT	Vendor ID	
				Product_type	UINT	Device type	
				Product_code	UINT	Product code	
				Major_rev	USINT	Major revision	
				Minor_rev	USINT	Monor revision	

Attribute ID	Need in implem	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
0x0C	Conditional ^a	Get/Set	NV	Connection disable	BOOL	Indicates if this instance of the Connection Configuration object is disabled	0 – This instance of the Connection Configuration object is enabled. 1 – This instance of the Connection Configuration object is disabled. When an Open service is received this value shall be set to 0. When a Close or Stop service is received this value shall be set to 1
0x0D	Conditional ^b			Safety Parameters	See IEC 61784-3-2		
0x0E	Conditional ^b			Safety Connection Parameter CRC			
0x0F	Conditional ^b			Safety Configuration Instance			
0x10	Conditional ^b			Safety ID Allocation			
0x11	Conditional ^b			Safety Target Connection Serial Number			

^a Attribute 0x0C shall be supported if conditional services Open_Connection (0x4C) and Close_Connection (0x4D) are supported. Attribute 0x0C shall not be supported if conditional services Open_Connection (0x4C) and Close_Connection (0x4D) are not supported.

^b These attributes are not allowed if the device is not a safety device. If the device is a safety device, see IEC 61784-3-2.

7.8.4.2 Connection status

The values for the originator connection status are defined in Table 78.

Table 78 – Originator connection status values

General Status	Extended status	Error Description
0x00	N/A	Connection is open
0x01	0x0000 or greater	See Connection Manager service request error codes, IEC 61158-6-2
0x02 – 0x26	0x0000 or greater	See General Status codes, IEC 61158-6-2
0xD0	0x0001	Connection is closed or stopped (via the Close or Stop services)
	0x0002	Connection open is pending
	0x0003	Connection close is pending

The values of the target connection status are defined in Table 79.

Table 79 – Target connection status values

General Status	Extended status	Error Description
0x00	0x0000	Connection is open
0x01	0x0001 or greater	The number of connection to this target
0xD0	0X0001	Connection is closed or stopped (via the Close or Stop services)

7.8.4.3 Connection flags

The bit patterns for the connection flags are defined Table 80.

Table 80 – Connection flags

Bit	Meaning
0	Connection 0 = Originator 1 = Target
1 – 3	O->T Real time transfer format 000 – Use 32-bit Run/Program header to indicate idle mode 001 – Use zero data length packet to indicate idle mode 010 – None. Connection is pure data and is modeless 011 – Heartbeat 100 – Reserved 101 – Safety (see IEC 61784-3-2) 100 thru 111 – reserved for future use
4 - 6	T->O Real time transfer format 000 – Use 32-bit Run/Program header to indicate idle mode 001 – Use zero data length packet to indicate idle mode 010 – None. Connection is pure data and is modeless 011 – Heartbeat 100 – Reserved 101 – Safety (see IEC 61784-3-2) 100 thru 111 – reserved for future use
7 - 15	Reserved

7.8.4.4 Target device ID

This attribute is the identity of the target device for this connection instance. This identity information is not used for verifying (keying) the online target device, it is used by a configuration tool to locate the correct electronic data sheet for the connection configuration described in this CCO instance. For Target instances this attribute shall be the identity of the device containing this CCO object.

7.8.4.5 CS data index number

The CS Data Index Number attribute is required for a device on CP 2/1; otherwise this attribute shall not be implemented.

The CS Data Index Number is a value set by the configuration software. This is the connection_index value returned from the Scheduling object read service (see 7.4.4 for a the definition of the Scheduling object services). This attribute is ignored for target instances.

7.8.4.6 Net connection parameters

7.8.4.6.1 conn_timeout

The conn_timeout is the value used for the connection timeout multiplier field defined in IEC 61158-6-2 (Forward_Open request). conn_timeout is ignored for target instances.

7.8.4.6.2 xport_class_and_trig

The xport_class_and_trig is the value used for the Transport Type/Trigger field defined in IEC 61158-6-2 (Forward_Open request). xport_class and_trig is ignored for target instances.

7.8.4.6.3 rpi_OT

The rpi_OT is the value used for the O_to_T RPI field defined in IEC 61158-6-2 (Forward_Open request). rpi_OT is ignored for target instances.

7.8.4.6.4 net_OT

The net_OT is the value used for the O_to_T Network Connection Parameters field in IEC 61158-6-2 (Forward_Open request). Only the size field is used for target instances.

7.8.4.6.5 rpi_TO

The rpi_TO is the value used for the T_to_O RPI field defined in IEC 61158-6-2 (Forward_Open request). rpi_TO is ignored for target instances.

7.8.4.6.6 net_TO

The net_TO is the value used for the T_to_O Network Connection Parameters field in IEC 61158-6-2 (Forward_Open request). Only the size field is used for target instances.

7.8.4.7 Connection path

7.8.4.7.1 open_path_size

The open_path_size is the value used for the Connection_Path_size field in IEC 61158-6-2 (Forward_Open request). For target instances the open_path_size is used for matching the open_path_size received in a forward open.

7.8.4.7.2 open connection path

The open_connection_path is the value used for the Connection_Path field in IEC 61158-6-2 (Forward_Open request). For target instances the open_connection_path is used for matching the open_connection_path received in a forward open.

7.8.4.8 Config #1 data

This does not apply to target instances. The data specified in attributes 0x07 and 0x0A are concatenated (in that order) into a single data segment, then appended to the connection path in attribute 0x06 to form the complete path sent in the Forward_Open service of the Connection Manager object.

All the configuration data may be placed in attribute 7 or all the configuration data may be placed in attribute 0x0A or the configuration data may be split between attributes 0x07 and 0x0A. When a connection is a proxied connection the convention that shall be followed is that

any configuration data intended for the proxying device is placed in attribute 0x07 and any configuration data intended for the proxied device is placed in attribute 0x0A. Config #1 data is ignored for target instances.

7.8.4.9 Connection name

This Connection Name field allows a user to name each connection instance. The connection name has no meaning to the Connection Configuration object.

7.8.4.10 I/O mapping

The I/O mapping attribute contains I/O mapping data. The I/O mapping data specifies image table locations where target to originator data is placed and where originator to target data is obtained.

Table 81 describes the structure of the mapping_data field of the I/O mapping attribute for each of the format numbers.

Table 81 – I/O mapping formats

Format Number	Mapping_data_size (in octets)	Name	Data Type	Description
0	4	Single I/O tables	STRUCT of 4 octets	16 bit words, 0 based 16 bit word offsets
		O->T offset	UINT	Offset into O->T image table
		T->O offset	UINT	Offset into T->O image table
1	8	Multiple I/O tables	STRUCT of 8 octets	Table selection, 16 bit words, 0 based 16 bit word offsets
		O->T table	UINT	O->T image table selection
		O->T offset	UINT	Offset into O->T image table
		T->O table	UINT	T->O image table selection
		T->O offset	UINT	Offset into T->O image table

7.8.4.11 Config #2 data

This note does not apply to target instances. The data specified in attributes 0x07 and 0x0A are concatenated (in that order) into a single data segment, then appended to the connection path in attribute 0x06 to form the complete path sent in the Forward_Open service of the Connection Manager object. All the configuration data may be placed in attribute 0x07 or all the configuration data may be placed in attribute 0x0A or the configuration data may be split between attributes 0x07 and 0x0A. When a connection is a proxied connection the convention that shall be followed is that any configuration data intended for the proxying device is placed in attribute 0x07 and any configuration data intended for the proxied device is placed in attribute 0x0A. Config #2 data is ignored for target instances.

7.8.4.12 Proxy Device ID

This attribute is the identity of the device proxying for the target device for this connection instance. This identity information is not used for verifying (keying) the online target device, it is used by a configuration tool to locate the correct electronic data sheet for the connection configuration described in this CCO instance. For Target instances and Connections that are not proxied this attribute shall be ignored.

7.8.5 Common services

7.8.5.1 General

The Connection Configuration object shall support the common services as specified in Table 82.

Table 82 – Connection configuration object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Required	Required	Get_Attribute_All	Gets all attributes of the specified instance
0x02	Optional	Required	Set_Attribute_All	Sets all attributes of the specified instance
0x08	Required	N/A	Creat	Creates a new connection instance
0x09	Required	Required	Delete	Deletes an existing connection instance
0x0E	Optional	Required	Get_Attribute_Single	Returns the contents of the specified attribute
0x10	Required	Optional	Set_Attribute_Single	Modifies an attribute value
0x15	Required	Required	Restore	Restore current connection attributes

7.8.5.2 Get_Attribute_All

This service shall read all attributes associated with an existing instance and shall support the error codes shown in Table 83.

Table 83 – Get_Attribute_All error codes

General Status	Extended status	Error Description
0x04	None	Path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service

The structure of the Get_Attribute_All response is shown in Table 84.

Table 84 – Get_Attribute_All response

Name	Data Type	Description
Connection status	STRUCT of 4 octets	Connection status information. When the connection is not open, this attribute contains an error code indicating the reason
	USINT	General status
	USINT	Pad
	UINT	Extended status
Connection flags	WORD	Connection flags
Target device id	STRUCT of 8 octets	Target device identification
	UINT	Vendor ID
	UINT	Product type
	UINT	Product code
	USINT	Major revision
	USINT	Minor revision
CS data index number	UDINT	Scheduling object read data connection index number. The default value used when this attribute is not supported shall be 0xFFFFFFFF
Net connection parameters	STRUCT of 14 octets	Network connection parameters for both originator-to-target and target-to-originator directions
	USINT	Connection time out multiplier
	SWORD	Transport class and trigger
	UDINT	Originator-to-target RPI
	UINT	Originator-to-target network connection parameters
	UDINT	Target-to-originator RPI
	UINT	Target-to-originator network connection parameters
Connection path	STRUCT of variable size	Connection path
	USINT	Size of connection path, in 16-bit words, used in the Connection Manager Forward_Open service request
	USINT	Reserved. Shall be zero
	ARRAY of UINT	Connection path. The open connection path size is the length of this array
Config #1 data	STRUCT of variable size	Config #1 data. This data is sent to devices via the Connection Manager Forward_Open service request
	UINT	Configuration data length in octets
	ARRAY of USINT	Configuration data padded to an even number of octets
Config #2 data	STRUCT of variable size	Config #2 data. This data is sent to devices via the Connection Manager Forward_Open service request
	UINT	Configuration data length in octets
	ARRAY of USINT	Module configuration data padded to even number of octets
Connection name	STRUCT of variable size	Connection name
	USINT	Number of characters in connection name
	USINT	Pad
	STRING2	Connection name encoded in UNICODE

Name	Data Type	Description
I/O mapping	STRUCT of variable size	I/O mapping
	UINT	Format number
	UINT	Attribute size in octets
	ARRAY of USINT	Attribute data padded to an even number of octets
Proxy device id	STRUCT of 8 octets	Proxy device identification
	UINT	Vendor ID
	UINT	Product type
	UINT	Product code
	USINT	Major revision
	USINT	Minor revision
Safety parameters	STRUCT of 55 octets	See IEC 61784-3-2. These 55 octets are included in the Get_Attribute_All response only if either the O=>T or T=>O real time transfer format in the Connection Flags Attribute (attribute 2) indicates the format is Safety
Connection parameter CRC		
Safety configuration instance		
Safety id allocation flag		
Safety target connection serial number		
Connection disable	BOOL	Bit 0, value of connection disable attribute (attribute 12) if supported. 0, if connection disable attribute is not supported

7.8.5.3 Set_Attribute_All

This service shall read all attributes associated with an existing instance and shall support the error codes shown in Table 85.

Table 85 – Set_Attribute_All error codes

General Status	Extended status	Error Description
0x04	None	Path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service
0x0C	None	Wrong object state

The structure of the Set_Attribute_All request is shown in Table 86.

Table 86 – Set_Attribute_All request

Name	Data Type	Description
Connection flags	WORD	Connection flags
Target device id	STRUCT of 8 octets	Target device identification
	UINT	Vendor ID
	UINT	Product type
	UINT	Product code
	USINT	Major revision
	USINT	Minor revision
Cs data index number	UDINT	Scheduling object read data connection_index number. The default value used when this attribute is not supported shall be 0xFFFFFFFF
Net connection parameters	STRUCT of 14 octets	Network connection parameters for both O⇒T and T⇒O directions.
	USINT	Connection time-out multiplier
	SWORD	Transport class and trigger
	UDINT	O⇒T RPI
	UINT	O⇒T network connection parameters
	UDINT	T⇒O RPI
	UINT	T⇒O network connection parameters
Connection path	STRUCT of variable size	Connection path
	USINT	Size of connection path, in 16-bit words, used in the Connection Manager Forward_Open service request
	USINT	Reserved. Shall be zero.
	ARRAY of UINT	Connection path. The open connection path size is the length of this array
Config #1 data	STRUCT of variable size	Config #1 data. This data is sent to devices via the Connection Manager Forward_Open service request
	UINT	Module configuration data length in octets.
	ARRAY of USINT	Module configuration data padded to even number of octets
Config #2 data	STRUCT of variable size	Config #2 data. This data is sent to devices via the Connection Manager Forward_Open service request
	UINT	Module configuration data length in octets
	ARRAY of USINT	Module configuration data padded to even number of octets
Connection name	STRUCT of variable size	Connection name
	USINT	Number of characters in connection name
	USINT	Pad.
	STRING2	Connection name encoded in UNICODE.
I/O mapping	STRUCT of variable size	I/O mapping
	UINT	Format number.
	UINT	Attribute size in octets

Name	Data Type	Description
	ARRAY of USINT	Attribute data padded to an even number of octets
Proxy device id	STRUCT of 8 octets	Proxy Device identification.
	UINT	Vendor Id
	UINT	Product type
	UINT	Product code
	USINT	Major revision
	USINT	Minor revision
Safety parameters	STRUCT of 49 octets	See IEC 61784-3-2. These 49 octets are included in the Set_Attribute_All service data only if either the O=>T or T=>O real time transfer format in the Connection Flags Attribute (attribute 2) indicates the format is Safety
Safety configuration instance		
Safety id allocation flag		
Connection disable	BOOL	Bit 0, value of connection disable attribute (attribute 12)

7.8.5.4 Create

This service creates a new instance of a Connection Configuration object. Initial attribute values may also be specified with this service. The created instance number is assigned by the class and returned to the requestor. Table 87 defines the request parameters for this service.

Table 87 – Create request parameters

Parameter	Data Type	Description
Connection flags	WORD	Connection flags
NumConnParams	UINT	Number of attribute/value pairs that follow
ConnParams	ARRAY of	List of Attribute number/value pairs
	STRUCT of variable size	
	UINT	Attribute number
	Object/class attribute specific STRUCT	Attribute value

This service shall read all attributes associated with an existing instance and shall support error codes shown in Table 88.

Table 88 – Create error codes

General Status	Extended status	Error Description
0x02	0x0001	Insufficient resource — maximum number of instances already exist
	0x0002	Insufficient resource — not enough memory on device
0x03	None	Invalid parameter error (when attribute count is invalid)
0x04	None	Path syntax error
0x0C	None	Wrong object state
0x0E	None	Attempt to set a read-only attribute
0x13	None	Insufficient request data — request was too short or truncated
0x1C	None	Attribute list shortage — required attribute was missing

7.8.5.5 Delete

This service deletes existing connection instances. If addressed to the class-level, all connection instances are deleted. If addressed to the instance-level, only the addressed instance is deleted. This service shall support the error codes shown in Table 89.

Table 89 – Delete error codes

General Status	Extended status	Error Description
0x04	None	Path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service
0x0C	None	Wrong object state

7.8.5.6 Restore

This service shall discard modifications to instances that have not yet been committed by the Change_Complete service. If the Restore service is addressed to the class (instance 0), then pending modifications for all instances shall be discarded and the edit session shall be ended. If addressed to a specific instance, only modifications for that instance shall be discarded and the edit session shall not be ended.

This service shall support the error codes shown in Table 90.

Table 90 – Restore error codes

General Status	Extended status	Error Description
0x04	None	Path syntax error
0x05	None	Instance undefined
0x0C	None	Wrong object state

7.8.6 Class specific services

7.8.6.1 General

The Connection Configuration object shall support the class specific services as specified in Table 91.

Table 91 – Connection configuration object class specific services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x4B	Required	N/A	Kick_Timer	Kicks Edit Watchdog Timer
0x4C	Conditional*	Conditional*	Open_Connection	Opens connections
0x4D	Conditional*	Conditional*	Close_Connection	Closes connections
0x4E	Conditional*	Conditional*	Stop_Connection	Stops connections
0x4F	Required	N/A	Change_Start	Manages session editing
0x50	Required	N/A	Get_Status	Get status for multiple connections
0x51	Required	N/A	Change_Complete	Completes session editing
0x52	Required	N/A	Audit_Changes	Audits pending changes
* The Open_Connection and Close_Connection services shall either both be supported or neither the Open_Connection or Close_Connection services shall be supported. The Stop_Connection service may only be supported if the Open_Connection service is supported				

7.8.6.2 Kick_Timer

This service shall reinitialize the edit watchdog timer. Upon successful execution of the Change_Start service, the edit watchdog timer shall be started with a period of 60 s. This timer is used to recover from the loss of a configuration client between the Change_Start and Change_Complete/Restore operations. Receipt of any service request shall reset the edit watchdog timer. Clients may request this service to reset the timer without otherwise affecting the state of the Connection Configuration object. If the edit watchdog timer expires, all pending modifications shall be discarded and the edit session shall be ended.

This service shall support the error codes shown in Table 92.

Table 92 – Kick_Timer error codes

General Status	Extended status	Error Description
0x04	None	Path syntax error
0x05	None	Instance undefined

7.8.6.3 Open_Connection

The Open_Connection service shall cause the connection associated with an instance of the Connection Configuration object to open. If the Open_Connection service is addressed to the class (instance 0), then all connection instances shall be opened. If addressed to a specific instance, then only that connection instance shall be opened.

This service shall support the error codes shown in Table 93.

Table 93 – Open_Connection error codes

General Status	Extended status	Error Description
0x04	None	Path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service

7.8.6.4 Close_Connection

The Close_Connection service shall cause the connection associated with an instance of the Connection Configuration object to close. If the Close_Connection service is addressed to the class (instance 0), then all connection instances shall be closed. If addressed to the instance level, then only the specified instance shall be closed. The Close_Connection service shall initiate a “graceful” connection shutdown, that is, a Forward_Close request shall be sent to the connection target. Once a connection has been closed by this service it shall remain closed until an Open_Connection service is issued.

This service shall support the error codes shown in Table 94.

Table 94 – Close_Connection error codes

General Status	Extended status	Error Description
0x04	None	Path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service

7.8.6.5 Stop_Connection

The Stop_Connection service shall cause the connection associated with an instance of the Connection Configuration object to stop producing data immediately without sending an Forward_Close request to the connection target. If the Stop_Connection service is addressed to the class (instance 0), then all connection instances shall be stopped. If addressed to the instance level, then the specified instance shall be stopped. Once a connection has been stopped, it remains stopped until a subsequent Open_Connection service request is issued.

This service shall support the error codes shown in Table 95.

Table 95 – Stop_Connection error codes

General Status	Extended status	Error Description
0x04	None	Path syntax error
0x05	None	Instance undefined
0x08	None	Unimplemented service

7.8.6.6 Change_Start

This service shall

- signal the beginning of an edit session;
- synchronize the current and pending attributes;

- c) place all connections in the "changeable" state;
- d) start the edit watchdog timer.

Change_Start shall be requested prior to performing any services that modify the attributes of a connection. This service shall only be addressed to the class-level (instance 0). If a Change_Start service is received while an edit session is active, an Object State Conflict error (0x0C) shall be returned.

This service shall support the error codes shown in Table 96.

Table 96 – Change_Start error codes

General Status	Extended status	Error Description
0x04	None	Path syntax error
0x0C	None	Wrong object state
0x10	None	Device state conflict

7.8.6.7 Get_Status

The Get_Status service shall retrieve the status attribute (attribute 1) for multiple connections via a single transaction. This service shall be supported at the class-level (instance 0) only. Given a starting instance number, the Get_Status service shall return instance/status pairs until either the response buffer is full or the status of all connections has been returned.

The request parameter are defined in Table 97.

Table 97 – Get_Status service parameter

Parameter	Data Type	Description
Starting Instance	UDINT	Starting instance number

The response parameters are defined for this service in Table 98.

Table 98 – Get_Status service response

Parameter	Data Type	Description
Done Indicator	UINT	0 = More status to be retrieved 1 = All connection status information has been retrieved. All other values reserved.
NumStatusEntries	UINT	Number of instance/status pairs that follow
StatusEntries	ARRAY of	List of instance/status pairs
	STRUCT of 8 octets	
	UDINT	General status
	USINT	General status
	USINT	Reserved, shall be 0
	UINT	Extended status

This service shall support the error codes shown in Table 99.

Table 99 – Get_Status service error codes

General Status	Extended status	Error Description
0x03	None	Invalid parameter error (when attribute count is invalid)
0x04	None	Path syntax error
0x08	None	Unimplemented service

7.8.6.8 Change_Complete

This service shall signal the completion of an edit session. Pending attributes for all modified connection instances shall be applied. This service shall take a parameter indicating the type of change being performed; either full or incremental. If an incremental edit is specified, then only the connections that have been modified shall be broken and re-established. If a full edit is specified, then all connections shall be broken and all supporting resources shall be freed and reallocated before attempting to re-establish the connections. The Change_Complete service shall be supported at the class-level (instance 0) only.

If optional instance attribute 12 is supported and the value of instance attribute 12 is FALSE or if optional instance attribute 12 is not supported an attempt to open the connection shall be made. If optional instance attribute 12 is supported and the value of instance attribute 12 is TRUE, no attempt shall be made to open the connection.

The request parameter is defined in Table 100.

Table 100 – Change_Complete service parameter

Parameter	Data Type	Description
Change type	UINT	0 = Full 1 = Incremental All other values reserved.

This service shall support the error codes shown in Table 101.

Table 101 – Change_Complete service error codes

General Status	Extended status	Error Description
0x02	None	Resource unavailable. Indicates that there is not enough memory on the host device to support the specified configuration
0x04	None	Path syntax error
0x0C	None	Wrong object state
0x10	None	Device state conflict

7.8.6.9 Audit_Changes

This service shall verify whether or not there is enough memory on the host device to support a proposed configuration. Like the Change_Complete service, this service shall take a parameter indicating the type of change being performed; either full or incremental. The Audit_Changes service shall be supported at the class-level (instance 0) only. This service allows a configuration client to determine if all pending changes will be successful before actually committing the changes with the Change_Complete service.

The request parameter is defined in Table 102.

Table 102 – Audit_Changes service parameter

Parameter	Data Type	Description
Change type	UINT	0 = Full 1 = Incremental All other values reserved

This service shall support the error codes shown in Table 103.

Table 103 – Audit_Changes service error codes

General Status	Extended status	Error Description
0x02	None	Resource unavailable. Indicates that there is not enough memory on the host device to support the specified configuration
0x04	None	Path syntax error
0x0C	None	Wrong object state
0x10	None	Device state conflict

7.8.7 Behavior

Figure 22 summarizes the process of creating, editing, and deleting connections.

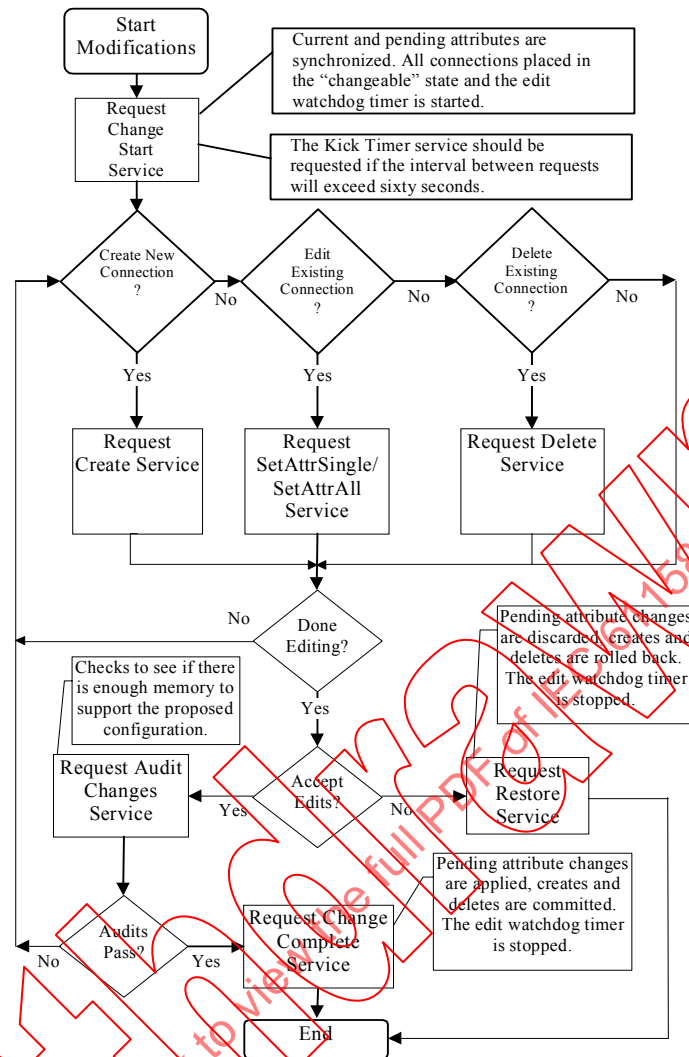


Figure 22 – Connection configuration object edit flowchart

8 Other DLE elements of procedure

8.1 Network attachment monitor (NAM)

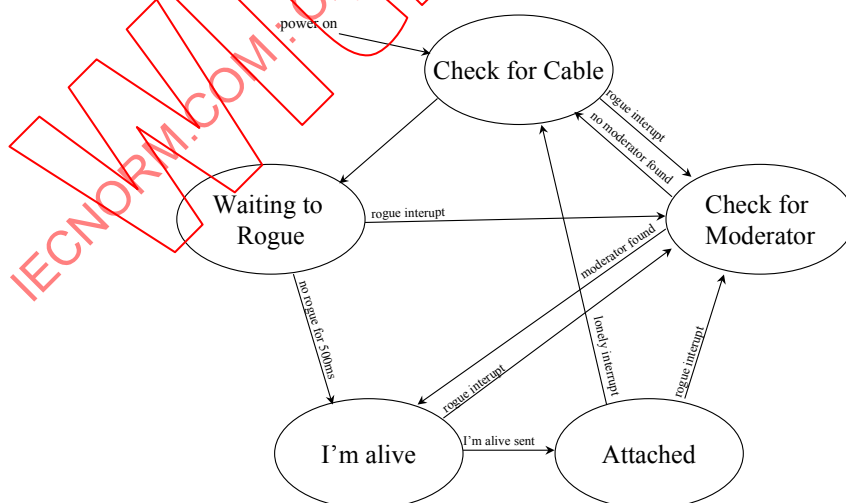
8.1.1 General

The network attachment monitor (NAM) shall control the DLL to allow new nodes to non-disruptively join a working link.

The NAM controls attaching the DLL to an existing link without causing disruption to transmissions on that link. It accomplishes this by monitoring and controlling the DLL via the Station Management and the normal send/receive interfaces to the DLL as summarized in Table 104 and Figure 23.

Table 104 – NAM states

State	Actions
Check for Cable	1) auto-address nodes shall find an address before entering (see 8.1.3) 2) use default parameters (see 8.1.2) 3) transmit DLPDUs, but no moderators 4) if receive rogue moderator, go to “Check for Moderator” 5) if receive any good DLPDU, transmit one more DLPDU, then go to “Wait to Rogue”
Wait to Rogue	1) use default parameters (see 8.1.2) 2) transmit no DLPDUs 3) if hear rogue moderator, go to “Check for Moderator” 4) if hear no rogue moderator for 400 – 600 ms, go to “I’m Alive”
Check for Moderator	1) transmit no DLPDUs 2) if hear 9 identical valid moderators in successive NUTs, go to “I’m Alive” using the parameters in those moderators 3) if have not heard a moderator for 1,0 s – 3,0 s, go to “Check for Cable” 4) if receive moderator with invalid link parameters, stay in this state, but change network status indicator to invalid link configuration (flashing red / green)
I’m Alive	1) auto-address nodes shall find an address before entering (see 8.1.3) 2) transmit DLPDUs, including moderators if this node has the lowest MAC ID 3) transmit three fixed tag 0x80 Lpackets 4) after sending them go to “Attached” 5) if receive rogue moderator, go to “Check for Moderator”
Attached	1) normal network operation 2) transmit DLPDUs, including moderators if this node has the lowest MAC ID 3) if receive rogue moderator, go to “Check for Moderator” 4) if no other nodes on link for 8 NUTs, go to “Check for Cable”

**Figure 23 – NAM state machine****8.1.2 Default parameters**

The default link parameters for a node shall be as shown in Table 105:

Table 105 – Default link parameters

Parameter	Data type	Value
NUT_length	UINT	100 ms (10 000 intervals of 10 µs each)
smax	USINT	0
umax	USINT	99
slotTime	USINT	254 (255 µs)
blanking	USINT	6 (octet times)
gb_start	USINT	610 µs (61 intervals of 10 µs each)
gb_center	USINT	450 µs (45 intervals of 10 µs each)
modulus	USINT	127
gb_prestart	USINT	920 µs (92 intervals of 10 µs each)

8.1.3 Auto-addressing

Some nodes may have no pre-assigned MAC ID. These nodes, known as auto-address nodes, shall search for an unused MAC ID by observing the link. The auto-address node shall not transmit until a free MAC ID is found.

NOTE Typically, auto-address nodes are transient nodes such as hand-held terminals.

8.1.4 Valid MAC IDs

On a link with default configuration parameters, an auto-address node shall search in the range 92 through 99. On a link with any other configuration parameters, an auto-address node shall search in the range SMAX+1 through UMAX. A free MAC ID shall be declared found if at least three sequential unscheduled transmit opportunities for that MAC ID have passed without receiving any DLPDUs from that MAC ID.

NOTE The ControlNet object provides an interface to determine the MAC ID which has been chosen.

8.1.5 State machine description

The following state machine description shall define the behavior of the network attachment monitor:

```
// Network Attachment Monitor state machine description
//
// This state machine monitors and controls the DLL to make sure that it goes on-line
// in a fashion that does not disrupt a running network.
//
//
// Lpacket constants: masks for ctl octet
//
#define FIXEDSCREEN 1
#define TAGPAD 2

#define DATAPAD 4
#define OCTETWORD 8
//
// moderator Lpacket constants
//
#define MODERATOR_SIZE 9           // moderator Lpacket size octet
#define MODERATOR_CTL 1           // moderator Lpacket control octet
#define MODERATOR_TAG 0           // moderator Lpacket tag octet

//
// Lpacket class
//
class Lpacket
{
public:
    USINT size;           // return the size octet
    USINT ctl;           // return the control octet
```

```

// return the value of the tag pad bit
int tag_pad(void)
{
    return (ctl&TAGPAD) >>1;
}
// return the value of the data pad bit
int data_pad(void)
{
    return (ctl&DATAPAD) >>2;
}
// return the number of octets in the Lpacket
int wire_size(void)
{
    return size*2 - tag_pad() - data_pad();
}
// get the next octet to be transmitted
USINT get_next_octet(void);

// is this Lpacket aborted?
BOOL abort(void);
};

class fixedLpacket : public Lpacket
{
public:
    USINT    tag;
    USINT    dest;
};

class moderatorLpacket : public fixedLpacket
{
public:
    UINT      NUT_length;
    USINT     smax;
    USINT     umax;
    USINT     slotTime;
    USINT     blanking;
    USINT     gb_start;
    USINT     gb_center;
    USINT     usr;
    USINT     interval_count;
    USINT     modulus;
    USINT     tMinus;
    USINT     gb_prestart;
    USINT     spare;

    BOOL isValid(void);
    // this checks for umax<myaddr, and similar parameter
    // error that would prevent a node from successfully
    // joining a network

    BOOL operator==(moderatorLpacket &);
    // the equality test for moderator Lpackets excludes
    // the interval count, and usr, but includes the
    // tMinus counter
};

//
// interface to station management
//
BOOL SM_powerup;           // input indication: powerup has occurred
void SM_LEDS(LED_STATE);  // set the LEDS to a specified pattern
#define LED_bad_network_parameters    // flashes the network status indicator

class DLL_config_data
{
public:
    BOOL    listen_only;
    USINT   myaddr;
    UINT     NUT_length;
    USINT    smax;
    USINT    umax;
    USINT    slotTime;
    USINT    blanking;
    USINT    gb_start;
    USINT    gb_center;
    USINT    interval_count;
};

```

```

USINT modulus;
USINT gb_prestart;
};

//
// interface to DLL
//
void DLL_xmit_fixed_request(char *comment);
void DLL_xmit_fixed_confirm(void);
void DLL_rcv_fixed_indication();

//
// These timer objects are as used internally to the ACM of the DLL.
//
// various behaviors for timers
enum {ONCE_UP, ONCE_DOWN, REPEAT_UP, REPEAT_DOWN} timer_mode;

class timer
{
public:

float count;           // number of ticks left
float preset;          // initialize ticks, or cycle length
BOOL expire;           // latched flag when timer expires
timer_mode mode;

// constructor: defines mode
timer(timer_mode m)
{
mode = m;
}

void restart(void)
{
if(mode == REPEAT_DOWN
|| mode == ONCE_DOWN) // if counting down, start with preset
{
count = preset;
}
else
{
count = 0;           // else start from zero
}
// start counting
}

void start(float interval) // start counting down from interval towards zero
{
mode = ONCE_DOWN;
preset = interval;    // set interval
restart();             // start counting
}

void begin_counting(void) // start counting up from zero
{
mode = ONCE_UP;
restart();
}

bool expired(void)       // returns true if the timer has expired; clears flag
{
BOOL tmp;
tmp = expire;
expire = 0;
return tmp;
}
};

//
// time conversion factors
//
#define usec
#define msec *1000
#define sec *1000000

class timer timer(ONCE_DOWN); // general purpose timer
int counter;                  // general purpose counter

```

```

moderatorLpacket new_mod;    // buffer for getting new moderator Lpackets
moderatorLpacket old_mod;    // buffer for saving moderator Lpackets

DLL_config_data default_config; // DLL config parameters for default mode
DLL_config_data DLL_config;    // scratch DLL config buffer

// Wait for powerup.
// Bring the DLL on-line.
// It is assumed that the node's MAC ID is known before this machine is allowed to
// power-up.
// If this node is auto-addressed, a free MAC ID shall be determined
// before enabling this machine.

state: powerup0

    event:    SM_powerup
    destination: checkForCable0
    action:
        DLL_set_current_request(default_config); // set default parameters

state: checkForCable0

    event:    DLL_set_current_confirm() // default parameters were set
    destination: checkForCable1
    action:
        DLL_enable_moderator_request(FALSE); // disable moderators
        DLL_online_request(TRUE); // send DLL on-line

state: checkForCable1

    event:    DLL_online_confirm() // DLL is now on-line, but can't moderate
    destination: checkForCable2

state: checkForCable2

    // rogue -> checkForModerator
    event:    DLL_SM_report(rogue)
    destination: checkForModerator0
    action:
        DLL_listen_only_request(TRUE); // send DLL to listen only
        DLL_enable_fixed_request(MODERATOR_TAG); // enable reception of moderators

    event:    DLL_rcv_fixed_indication
    || DLL_rcv_gen_indication // any Lpacket received
    destination: waitForRogue
    action:
        wait_until transmit once more;
        DLL_listen_only_request(TRUE); // send DLL to listen only
        DLL_enable_moderator_request(FALSE); // re-enable moderators
        timer.start(500 msec);

state: waitForRogue

    // rogue -> checkForModerator
    event:    DLL_SM_report(rogue)
    destination: checkForModerator0
    action:
        DLL_listen_only_request(TRUE); // send DLL to listen only
        DLL_enable_fixed_request(MODERATOR_TAG); // enable reception of moderators

    event:    timer.expired()
    destination: alive0
    action:
        DLL_listen_only_request(FALSE); // send DLL to listen only
        DLL_xmit_fixed_request("send an I'm alive message");

//
// in the "alive" states, the DLL is on-line
// three "I'm alive" message are sent
// if lonely is detected, do nothing
//
state: alive0

```

```

// rogue -> checkForModerator
event: DLL_SM_report(rogue)
destination: checkForModerator0
action:
    DLL_listen_only_request(TRUE); // send DLL to listen only
    DLL_enable_fixed_request(MODERATOR_TAG); // enable reception of moderators

// first message sent, send the second
event: DLL_xmit_fixed_confirm();
destination: alive1
action:
    DLL_xmit_fixed_request("send an I'm alive message");

state: alive1

// rogue -> checkForModerator
event: DLL_SM_report(rogue)
destination: checkForModerator0
action:
    DLL_listen_only_request(TRUE); // send DLL to listen only
    DLL_enable_fixed_request(MODERATOR_TAG); // enable reception of moderators

// second message sent, send the third
event: DLL_xmit_fixed_confirm();
destination: alive2
action:
    DLL_xmit_fixed_request("send an I'm alive message");

state: alive2

// rogue -> checkForModerator
event: DLL_SM_report(rogue)
destination: checkForModerator0
action:
    DLL_listen_only_request(TRUE); // send DLL to listen only
    DLL_enable_fixed_request(MODERATOR_TAG); // enable reception of moderators

// third message send, consider attachment to be complete
event: DLL_xmit_fixed_confirm();
destination: attached

//
// the DLL is on-line and fully functional
//
state: attached

// rogue -> checkForModerator
event: DLL_SM_report(rogue)
destination: checkForModerator0
action:
    DLL_listen_only_request(TRUE); // send DLL to listen only
    DLL_enable_fixed_request(MODERATOR_TAG); // enable reception of moderators

event: DLL_SM_report(lonely)
destination: checkForCable
action:
    DLL_enable_moderator_request(FALSE); // disable moderators

//
// a rogue event was detected
// the DLL instantly disables itself (it goes to its rogue state)
//
state: checkForModerator0

event: DLL_enable_fixed_confirm(); // receive moderators enabled
destination: checkForModerator1
action:
    counter = 0; // init moderator counter
    old_mod = NULL;
    timer.start(3 sec);

state: checkForModerator1

// timeout
event: timer.expired
destination: checkForCable0
action:
    DLL_set_current_request(default_params); // set default parameters

// ignore irrelevant received DLPDUs
event: DLL_recv_fixed_indication(new_mod) // got a moderator Lpacket
condition: received Lpacket is not mod
    || received Lpacket is not TUI
    || TUI.net_change == false
destination: checkForModerator1

```

```

// received a TUI indicating a net change in progress -- reset counter
event: DLL_rcv_fixed_indication(new_mod) // got a moderator Lpacket
condition: received Lpacket is TUI
           && TUI.net_change == TRUE // the new moderator is different
destination: checkForModerator1
action:
    counter = 0;

// received a moderator that would not permit this node
// to operate normally (for example, - myaddr>umax)
// reset counter and flash bad_net_parameter indicator
event: DLL_rcv_fixed_indication(new_mod) // got a moderator Lpacket
condition: !isValid(new_mod) // the new moderator is no good for us
destination: checkForModerator1
action:
    SM_LEDS(LED_bad_network_parameters); // flash the indicator
    old_mod = new_mod;
    counter = 0;

// received moderator that doesn't match the last one -- reset counter
event: DLL_rcv_fixed_indication(new_mod) // got a moderator Lpacket
condition: received Lpacket is mod
           && new_mod != old_mod // the new moderator is different
destination: checkForModerator1
action:
    old_mod = new_mod;
    counter = 0;

// received moderator that matches, but not the ninth match -- just increment the
counter
event: DLL_rcv_fixed_indication(new_mod) // got a moderator Lpacket
condition: received Lpacket is mod
           && isValid(new_mod)
           && new_mod==old_mod
           && counter<8
destination: checkForModerator1
action:
    counter++;

// have received 9 successive identical moderators
// adopt these parameters and proceed to go back on-line
event: DLL_rcv_fixed_indication(new_mod) // got a moderator Lpacket
condition: received Lpacket is mod
           && isValid(new_mod)
           && new_mod==old_mod
           && counter==8
destination: checkForModerator2
action:
    DLL_online_request(FALSE); // send DLL off-line

//
// second step to going back on-line
//
state: checkForModerator2

event: DLL_online_confirm() // now off-line
destination: checkForModerator3
action:
    copy parameters from new_mod to DLL_config; // get net configuration from
moderator
    DLL_set_current_request(DLL_config); // adopt those parameters

//
// third step to going back on-line
//
state: checkForModerator3

event: DLL_set_current_confirm() // have now adopted new parameters
destination: alive0
action:
    DLL_listen_only_request(FALSE); // send DLL to listen only
    DLL_xmit_fixed_request("send an I'm alive message");

```

8.2 Calculating link parameters

8.2.1 Link parameters

This subclause describes the requirements governing the selection of values for the following configuration variables:

- NUT_length;
- gb_prestart;

- gb_start;
- gb_center;
- slotTime;
- blanking.

NOTE The definitions of the link parameters include NUT_length, gb_center, gb_prestart, gb_start, blanking, and slotTime.

8.2.2 Conditions affecting link parameters

Parameter value selection shall allow for worst case conditions of these factors:

- signal propagation time between nodes;
- the timing reference accuracy at various nodes;
- moderator changes resulting in a change in the cable distance or signal propagation time between moderator and non-moderator nodes.

NOTE 1 A typical moderator change occurs as follows: At some random time the previous moderator is disconnected from the cable, loses power, or stops transmitting moderators for some other reason. Two NUTs (sometimes three) pass during which no moderator DLPDUs are transmitted. The new moderator transmits its first moderator DLPDU in the guardband of the third NUT since it last received a valid moderator DLPDU.

NOTE 2 In the event that the previous moderator was lost near the time that the moderator DLPDU was being transmitted it may be possible that the new moderator received the last moderator DLPDU from the previous moderator, while other nodes may not have received the last moderator DLPDU. It is therefore required that nodes tolerate an interval of 4 NUTs between reception of valid moderator DLPDUs.

8.2.3 Moderator change

In the requirements which follow, the term “moderator change” shall be interpreted by a non-moderator node as:

- reception of a valid moderator DLPDU from the original moderator node
- a time interval of up to 4 NUTs during which no valid moderator DLPDUs are received
- reception of a valid moderator DLPDU from a second moderator node

8.2.4 NUT timing

8.2.4.1 Tone

A network update time (NUT) for a node shall begin at a point in time called the tone, and shall end at the next tone.

NOTE The times specified in this subclause assume a perfectly accurate clock. Any permitted inaccuracy is explicitly included in the requirements. Sources of inaccuracy may include firmware delay, hardware delay, and local crystal inaccuracy. Since some of the time range requirements do not explicitly specify the clock accuracy tolerance, implementations should provide that internal timers are set such that the requirements are met.

8.2.4.2 Clock accuracy

The timing specifications for PhL Signalling to support the tone and NUT precision shall be as defined in Table 106.

Table 106 – PhL timing characteristics

Specification	Limits / characteristics	Comments
Data Rate	5 Mbit/s $\pm$ CA	also called M_Symbol rate, <i>data “zero” or “one”</i>
Bit Time	200 ns $\pm$ CA	also called M_Symbol time, <i>data “zero” or “one”</i>
PhL symbol time	100 ns $\pm$ CA	also called Phy_Symbol time, see data encoding rules
Clock Accuracy (CA)	$\pm$ 150 parts/million maximum	including temperature, long term, and short term stability

8.2.4.3 Restriction on NUT_length

The maximum value of NUT_length shall be 10 000 (100 ms). The minimum value NUT_length shall ensure that each NUT allows a maximum length unscheduled Lpacket to be sent after the scheduled Lpackets have been sent.

NOTE For example, with no scheduled connections, the minimum value of NUT_length is about 1 ms.

8.2.4.4 Moderator node timing

The moderator node shall begin a new NUT at $(\text{NUT\_length} \times 10 \mu\text{s}) \pm \text{CA}$ after the previous tone. The reception of a valid or invalid DLPDU shall not affect the timing of the transmission of the moderator DLPDU by the moderator node.

The moderator node shall begin transmission of the moderator DLPDU at $(\text{NUT\_length} - \text{gb\_center}) \times 10 \mu\text{s} \pm \text{CA}$ after the tone.

8.2.4.5 Non-moderator node timing

A non-moderator node shall begin a new NUT at $(\text{gb\_center} \times 10 - 40 \pm 2) \mu\text{s}$ after the last M_Symbol of the end delimiter of any valid received moderator DLPDU. A non-moderator node shall begin a new NUT at $(\text{NUT\_length} \times 10) \mu\text{s} \pm \text{CA}$ after the previous tone if a valid moderator DLPDU is not received in the current NUT.

Every non-moderator node shall receive and process any valid moderator DLPDU which begins between $(\text{tone} + 20) \mu\text{s}$ and $(\text{tone} + (\text{NUT\_length} \times 10 - 30) \mu\text{s} \pm \text{CA})$.

8.2.4.6 Post-tone silence

No DLPDUs shall be transmitted between the tone and $20 \mu\text{s}$ after the tone.

8.2.4.7 Pre-tone silence

No DLPDU shall be transmitted, except the moderator DLPDU, that does not terminate before $((\text{NUT\_length} - \text{gb\_prestart}) \times 10 + 11,2) \mu\text{s} \pm \text{CA}$ after the tone.

The value of gb_prestart shall provide that the blanking time after the last non-moderator DLPDU transmitted by any node and received by any other node expires before the start of the guardband in the receiving node. This condition shall also be met during a moderator change.

8.2.4.8 Guardband start

The guardband shall start at $(\text{NUT\_length} - \text{gb\_start}) \times 10 \mu\text{s} \pm \text{CA}$ after the tone.

The value of gb_start shall provide that any node shall observe the start of the moderator DLPDU no earlier than $((\text{NUT\_length} - \text{gb\_start}) \times 10 + 20) \mu\text{s} \pm \text{CA}$ after the previous tone in that node. This condition shall be met during a moderator change.

8.2.4.9 Guardband center

The value of gb_center shall provide that any node shall observe the end of the moderator DLPDU no later than $(\text{NUT\_length} \times 10 - 40) \mu\text{s} \pm \text{CA}$ after the tone. This condition shall also be met during a moderator change.

8.2.4.10 Useable time

Every node shall receive and process any valid DLPDU that begins between $(\text{tone} + 20 \mu\text{s})$ and $(\text{tone} + (\text{NUT\_length} - \text{gb\_start}) \times 10 \mu\text{s} \pm \text{CA})$, except that reception of a valid or invalid DLPDU shall not affect the timing of the transmission of the moderator DLPDU by the moderator node.

8.2.5 Slot timing

8.2.5.1 Slot time

In the following description, the slots shall be numbered sequentially. The value of `slotTime` shall provide that a DLPDU transmitted in slot N is received in slot N at all other nodes assuming worst case conditions of

- crystal drift;
- signal propagation delay between nodes;
- MAC ID assignment;

but excluding noise events.

8.2.5.2 First slot

Scheduled slot 0 shall begin at 20 µs after the tone.

8.2.5.3 Missing node

If no DLPDU is sent or received during slot N-1, slot N shall begin at $\text{slotTime} \times 1 \mu\text{s} \pm \text{CA}$ after the beginning of slot N-1.

8.2.5.4 Restarting the slot timer

If a DLPDU is sent or received during slot N-1, slot N shall begin no later than 6,0 µs after the last `M_Symbol` of the end delimiter of the DLPDU. If a damaged DLPDU, with `ph_status_indication` not equal to Normal, is received during slot N-1, slot N shall begin no later than 6,0 µs after `ph_lock_indication` goes FALSE.

8.2.5.5 Node turn around time

If both node N-1 and node N transmit in their respective slots, node N shall begin transmitting between 11,2 µs and 12,8 µs after the last `M_Symbol` of the end delimiter of the DLPDU transmitted by node N-1.

8.2.5.6 Beginning transmission after missed node

If node N-1 does not transmit in its slot, and node N is enabled to transmit in its slot, node N shall begin transmitting no earlier than the start of slot N, and no later than 6,8 µs after the start of slot N.

NOTE The major factor to consider here is crystal drift. In the worst case, where there are two nodes attached to the network at MAC IDs 0 and 99, `SMAX=98`, `UMAX=99`, and `USR=1`, there can be a silence of 196 slot times between the scheduled DLPDU transmitted by node 0 and the unscheduled DLPDU transmitted by node 99. If node 99 has a fast crystal and node 0 has a slow crystal, or vice versa, the value of `slotTime` should be adequate to guarantee that, when node 99 transmits, node 0 agrees that the current slot is 99.

8.2.6 Blanking

The value of `blanking` shall be 6.

8.2.7 Example implementation (informative)

The requirements in 8.2.4, 8.2.5 and 8.2.6 are met (not optimally) in the following example program which calculates the link parameters:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define roundup(x,y) (((x) - 1L) / (y) + 1L) /* Round up integer division */
```

```

#define F_TURN      16060L    /* ns for firmware turnaround and detect */
#define B_TURN      5720L    /* ns extra turnaround with large blanking value */
#define NULLFRAME   7L      /* octets for length of a null Lpacket transmission */
#define DELAY       4110L    /* ns per 1000 meters cable delay */
#define RPT_DLY     815L     /* ns per repeater delay */
#define PPMT        1500L    /* Max crystal error in tenths of PPM = +/- 150,0 ppm */

#define MODBEFORE    2       /* min moderator overhead before Lpacket (housekeeping) */
#define MODRECOV     3       /* worst case min moderator overhead after any Lpacket */
#define MODTIME      5       /* ticks for the moderator plus error recovery if bad */
#define MINUNSCHED   850     /* minimum unscheduled time */

int main(int argc, char **argv)
{
    /* Default values for command line input parameters. These are
     * arbitrarily chosen for our typical convenient network sizes.
     */
    int dest = 0xff;
    int board = -1; /* If -1 use driver, else use XT version TX */
    int redund = 3; /* Unspecified. Later code defaults to 3 */
    long NUT_time = 10000L;
    int length = 1000;
    int smax = 7;
    int umax = -1;
    int repeater = 0;
    int blank = 6;
    int maxfrm = 255;
    int intmod = 256;
    int terse = 0;
    int listenonly = 0;
    int noleds = 0;

    /* Other derived variables. */

    int gap;
    long drift;
    long prop;
    long turn;
    long bslot;
    long slot;
    long picodev;
    int center;
    int start;
    int prestart;
    long unscheduled;
    long nuttenths;
    int nuthigh;
    int nutlow;
    char txhead[7];
    char *a;

    /* Parse arglist */

    argc--;
    argv++;
    if(argc==0)
    {
        argc=1;
        argv[0]="h";
    }

    while(argc && (a=argv[0][0], *a++=='-'))
    {
        while(*a) switch(*a++)
        {
            case 'x' :
                board = atoi(a);
                if ((board < 0) || (board > 3))
                {
                    printf("%%ERROR -- 0 <= board number <= 3.\n");
                    return(1);
                }
                goto nextarg;

            case 'l':
                listenonly = 128;
                break;
            case 'n':
                noleds = 64;
                break;
            case 'b':
                redund = 1;
                break;
            case 'a':
                redund = 2;
                break;
            case 'r':
                redund = 3;

```

```

        break;
    case 't':
        terse = 1;
        break;

    case 'h' :
    default:
        printf("Compute and print a net config Lpacket.\n\n");
        printf("NETCONF [-abdhlrnt] [-x<board>] [0x<dest>]"
            " [<NUT time> [<length> [<smax>]\n");
        printf("          [<umax> [<repeaters> [<blanking> [<maxframe> "
            " [<int modulus>]]]]]]]\n");
        return(1);
    }
nextarg:  argc--;
        argv++;
    }

    /* Check for network address option */
    if (argc && argv[0][0] == '0' && argv[0][1] == 'x')
    {
        sscanf (argv[0], "%i", &dest);
        if ((dest < 0) || (dest > 255))
        {
            printf("%%ERROR -- 0x00 <= destination address <= 0xff.\n");
            return(1);
        }
        argc--;
        argv++;
    }

    /* Parse remaining arguments.
    */
    switch (argc)
    {
    case 8:  intmod = atoi(argv[7]);
    case 7:  maxfrm = atoi(argv[6]);
    case 6:  blank = atoi(argv[5]);
    case 5:  repeater = atoi(argv[4]);
    case 4:  umax = atoi(argv[3]);
    case 3:  smax = atoi(argv[2]);
    case 2:  length = atoi(argv[1]);
    case 1:  NUT_time = atoi(argv[0]);
    }

    /* check the validity of all the arguments */
    if ((intmod < 1) || (intmod > 256))
    {
        printf("%%ERROR -- 1 <= interval count modulus <= 256.\n");
        return(1);
    }
    if ((maxfrm < 0) || (maxfrm > 255))
    {
        printf("%%ERROR -- 0 <= max scheduled frame <= 255.\n");
        return(1);
    }
    if ((blank < 0) || (blank > 255))
    {
        printf("%%ERROR -- 0 <= blanking <= 255.\n");
        return(1);
    }
    if (repeater < 0)
    {
        printf("%%ERROR -- number repeaters >= 0.\n");
        return(1);
    }
    if ((smax < 0) || (smax > 254))
    {
        printf("%%ERROR -- 0 <= max scheduled addr <= 254.\n");
        return(1);
    }
    if (length < 0)
    {
        printf("%%ERROR -- cable length >= 0.\n");
        return(1);
    }
    if ((NUT_time < 350L) || (NUT_time > 100000L))
    {
        printf("%%ERROR -- 350 <= nut time <= 100,000.\n");
        return(1);
    }

    if (umax == -1) umax = smax;
    if ((umax < smax) || (umax > 254))
    {
        printf("%%ERROR -- smax <= umax <= 254.\n");
    }

```

```

    return(1);
}
if (NUT_time%10!=0)
{
    printf("%%ERROR -- NUT length is specified in 10 µs intervals.\n");
    return(1);
}

/* Gap is the number of slot times that can pass in the normal
protocol between transmissions.

When smax<umax, then the maximum silence equals smax + umax - 1.
The maximum silence occurs on the network with smax<umax when
the first node is at 0, the only other is at umax with the USR==1.

To illustrate a transmit sequence example when smax<umax,
set smax=4 and umax=5, USR=1, with two nodes at 0 and 5.
Slot times -> ssssuuuu -> (smax + umax - 1)
01234123450

When smax=umax, then the maximum silence equals smax + umax - 2.
The maximum silence occurs on the network with smax = umax when
the first node is at 0, the only other is at 1 with the USR==2.

To illustrate a transmit sequence example when smax=umax,
set smax=umax=5, USR=2, with two nodes at 0 and 1.
Slot times -> ssssuuuu -> (smax + umax - 2)
012345234501

Units: slot times
*/

gap = (smax < umax) : smax + umax - 1 : smax + umax - 2;
if (gap < 0) gap = 0;

/* the drift is a correction factor needed to compensate for
the amount of time that the fastest and slowest nodes drift
apart during the longest silence due to crystal inaccuracy.

Actual drift factor = (1-(2*gap+1)(ppm))/ (1-ppm^2)
however, ppm^2 is so small it may be ignored.

Units: unitless factor scaled by micro/pico
*/

drift = roundup(10000000L - (long)(2*gap+1) * PPMT , 10L);

/* Prop is the time the signal takes to travel down the cable and
work its way through all the repeaters and modems. It assumed
on even a cable system that has no repeater boxes, that two
two repeater exist to allow nodes connected via the NAP.

Units: ps
*/

prop = (long)(length)*DELAY + (long)(repeater+2)*RPT_DLY*1000L;

/* turn is:
- the amount of time it takes for the DLL to respond to a
transmission, that is, - the time from the end of the end delimiter
to the start of the preamble

plus
- the time it takes another DLL to detect that the first *has*
responded, that is, - the time from the start of the preamble to the
end of the start delimiter.

Units: ns
*/

/* turnaround could be */
turn = (long)(blank)*1600L + B_TURN; /* <-- blanking constrained */
if (turn < F_TURN) turn = F_TURN; /* <-- or firmware constrained */

/* bslot is the ideal slot time - which is usually defined as
2 props + turnaround + detect (note that in this case, the
detect time and the turn time are combined)
Units: ps
*/

bslot = 2*prop + (turn*1000L);

/* slot is the ideal slot time adjusted to account for variances in
crystal frequency.
Units: µs

```

```

*/
slot = roundup(bslot , drift);
if ((slot < 1L) || (slot > 256L))
{
    printf("%%ERROR -- Slot time is large, reduce network complexity. \n");
    return(1);
}

/* Calculate guardband parameters (round up to a multiple of a 10 µs tick clock).*/

/* calc nut time - units: 10 usec ticks */
nuttenth = roundup(NUT_time,10L);

/* picodev is the amount the clocks of the slowest and fastest
   node drifts apart during four NUTs
   Units: ps
*/
picodev = nuttenth * PPMT * 8L;

/* center is the time before the next tone that the moderator
   begins transmitting the moderator DLPDU. This includes adjustments
   for clock drift during up to 4 NUTs, plus a moderator switchover
   from a near to a far node.
   Unit: 10 usec ticks
*/
center = MODTIME + MODRECOV +
    (int)roundup(picodev + 2*prop , 10000000L);

/* start is the time before the next tone that the guardband starts.
   Unit: 10 usec ticks
*/
start = center + MODBEFORE +
    (int)roundup(picodev , 10000000L);

/* prestart is the time before the next tone beyond which nodes may not
   transmit. Due to clock skew and other factors a transmission which
   is sent such that it ends just at prestart may actually be received
   by another node such that the transmission ends just as the guardband
   starts. The prestart margin is required in order to ensure that
   transmissions from slow nodes do not cross over into the guardbands
   of faster nodes.
   Unit: 10 usec ticks
*/
prestart = (int)roundup((nuttenth*PPMT*2L) + ((NULLFRAME+blank)*1600000L)
    + bslot , 10000000L) + start;

/* Determine if nut parameters is practical and print warnings.
   * Initial calculations are conservative due to roundup of several
   * parameters. Usage of fixround helps correct it.
   */

/* calc the max length of unscheduled time assuming that all
   scheduled nodes are silent.
   Units: µs
*/
unscheduled = NUT_time - max(roundup(NULLFRAME*1600000L+prop+turn*1000L,
    1000000000L),slot)*(long)(smax+1)
    - (long)(prestart)*10L;

if (unscheduled < MINUNSCHED)
    printf("%%WARNING -- Not enough scheduled time available. Increase nut
time.\n");

/* Print statistics and the transmit command.
*/
nutlow = nuttenth & 0xff;
nuthigh = (nuttenth >> 8) & 0xff;
if(!terse)
{
    long deviation = roundup(picodev , 10000000L);

    printf("slot = %ld usec, max unscheduled time = %ld usec\n",
        slot,unscheduled);
    printf("dev = %ld usec, prestart = %d ticks, start = %d ticks, center = %d
ticks\n",
        deviation,prestart,start,center);
}

if (board == -1) sprintf(txhead,"tx");
else sprintf(txhead,"txt /%d",board);
printf( " %s m a 17 %02x %02x %02x %02x %02x %02x %02x "
    "%02x %02x %02x %02x %02x %02x 00 00 00\n",
    txhead,dest,nutlow,nuthigh,smax,umax,(int)slot-1,blank,
    start,center,(listenonly|noleds|redund),maxfrm,intmod-1,prestart);
return(0);
}

```

9 Detailed specification of DL components

9.1 General

This clause specifies each main functional component of the DL internal architecture as shown in Figure 2 using the modeling language of 6.1.

9.2 Access control machine (ACM)

The Access Control Machine (ACM) shall control the sequencing of transmissions by this node.

```
// ACM State Machine Description
//
// constant and type definitions
//
// generic type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;

//
// protocol constants
//
#define LOWCOUNTINIT      2 // NUTs as lowman before switchover to moderator
#define MODERATOR_LENGTH 40 // length of the moderator DLPDU in usec
#define TXDUPCHECKS        3 // moderators that have to be heard before transmit at
// powerup
#define TXNOMOD             5 // number of missed moderators before disabling transmit
#define DEAFNESS            5 // number of missed moderators before adjusting phase
#define DEAFADJUST         5 // 10 usec ticks to slip NUT if deafness is detected
#define LONELYINIT         8 // NUTs that are tolerated without hearing anything
// before becoming lonely (should be longer than DEAFNESS
// to ensure time for deaf recovery)
//
// Lpacket definition
//
// Lpacket constants: masks for control octet
//
#define FIXEDSCREEN        1
#define TAGPAD             2
#define DATAPAD            4

//
// moderator Lpacket constants
//
#define MODERATOR_SIZE     9 // moderator Lpacket size octet (in 16 bit words)
#define MODERATOR_CTL      1 // moderator Lpacket ctl octet (fixed tag)
#define MODERATOR_TAG      0 // moderator Lpacket service octet

//
// Lpacket class
//
class Lpacket
{
public:
    USINT size; // the size octet
    USINT ctl;  // the ctl octet

    // return the value of the tag pad bit
    int tag_pad(void)
    {
        return (ctl&TAGPAD) >>1;
    }

    // return the value of the data pad bit
    int data_pad(void)
    {
        return (ctl&DATAPAD) >>2;
    }

    // return the size of the Lpacket (in octets)
    int wire_size(void)
    {
        return size*2 - tag_pad() - data_pad();
    }

    // get the next octet to be transmitted
    USINT get_next_octet(void);
};
```



```

    // is this Lpacket aborted?
    BOOL abort(void);
};

//
// fixed tag Lpacket (two octet tag)
//
class fixedLpacket : public Lpacket
{
public:

    USINT service;
    USINT dest;
};

//
// moderator fixed tag Lpacket
//
class moderatorLpacket : public fixedLpacket
{
public:

    UINT NUT_length;
    USINT smax;
    USINT umax;
    USINT slotTime;
    USINT blanking;
    USINT gb_start;
    USINT gb_center;
    USINT usr;
    USINT interval_count;
    USINT modulus;
    USINT tMinus;
    USINT gb_prestart;
    USINT spare;
};

/////////////////////////////////////////////////////////////////
//
// internal VARIABLES
//
// unless otherwise specified, all variables are initialized to zero at powerup

int tmp;
USINT interval_count; // counts how many NUTs have passed (0 - modulus)
USINT itr; // implicit token register - tracks which MAC should xmit
USINT usr; // unscheduled start register -- first unscheduled token
USINT tMinus; // countdown to synchronized change
USINT lowcount; // how many times have I been lowman?
USINT modcount; // counts moderators (or missed moderators) depends on netState
USINT modcountinit; // how many NUTs to check for dupnode
USINT nglowman; // optimistic lowman detector
USINT qlowman; // pessimistic lowman detector
USINT lonely; // how many NUTs have I been lonely
USINT deafcount; // number of times moderator apparently occurred during deaf
period
BOOL in_guardband; // TRUE during the guardband
BOOL dupflag; // edge detector for dupnode
BOOL scheduled; // scheduled/unscheduled part of the NUT
UINT octetsleft; // octets / octets left before end of current transmitted DLPDU
USINT macSrc; // source MAC ID for the current DLPDU
USINT currentMod; // the current moderator
moderatorLpacket moderator; // buffer for moderator Lpackets

enum { OFFLINE,
    WATCH,
    DUPCHECK,
    NOTMOD,
    MOD,
    ROGUE,
    DUPNODE
} netState; // connection state

/////////////////////////////////////////////////////////////////
// timer definition
//
// various behaviors for timers
typedef enum {ONCE_UP, ONCE_DOWN, REPEAT_UP, REPEAT_DOWN} timer_mode;

class timer
{
public:
    float count; // number of ticks left
    float preset; // init ticks, or cycle length

```

```

    BOOL expire;          // latched flag when timer expires
    timer_mode mode;

    // constructor: defines mode
    timer(timer_mode m)
    {
        mode = m;
    }

    void restart(void)
    {
        // if counting down, start with preset
        if(mode == REPEAT_DOWN || mode == ONCE_DOWN)
        {
            count = preset;
        }
        else
        {
            count = 0;    // else start from zero
        }
        // start counting (implementation not specified)
    }

    void start(float interval) // start counting down from interval towards zero
    {
        mode = ONCE_DOWN;
        preset = interval; // set interval
        restart();          // start counting
    }

    void begin_counting(void) // start counting up from zero
    {
        mode = ONCE_UP;
        restart();
    }

    bool expired(void)        // returns true if the timer has expired; clears flag
    {
        BOOL tmp;

        tmp = expire;
        expire = 0;
        return tmp;
    }
};

//
// time conversion factors
//
#define usec
#define msec *1000
#define sec *1000000

//
// define the timers used by the ACM
//
class timer gen_timer(ONCE_DOWN); // general purpose, in several places
class timer slot_timer(REPEAT_DOWN); // used for response time-out between messages
class timer watch_timer(ONCE_DOWN); // used for watch time state
class timer NUT_timer(REPEAT_DOWN); // this generates the NUT timing
class timer holdoff_timer(ONCE_DOWN); // holds off transmit for a timer after transmit
or receive

////////////////////////////////////
//
// interface to PhL
//
BOOL ph_lock_indication; // optimistic DLPDU detect (clock recovery is tracking)
BOOL ph_frame_indication; // pessimistic DLPDU detect (Manchester valid since the SD)

////////////////////////////////////
//
// interface to Deserializer
//
BOOL RX_dataReady; // an octet of a DLPDU is available
USINT RX_rxData; // octet most recently received from DLPDU (source MAC ID)
BOOL RX_endMAC; // the end delimiter has been detected
BOOL RX_FCSOK; // FCS on last DLPDU was OK
BOOL RX_abort; // abort received on last DLPDU

```

```

/////////////////////////////////////////////////////////////////
//
// interface to RxM
//
BOOL RX_receivedLpacket; // an Lpacket has been received
moderatorLpacket RX_Lpacket; // the Lpacket most recently received

/////////////////////////////////////////////////////////////////
//
// interface to transmitter (TxM)
//
void TX_sendHeader(USINT myaddr); // send preamble, SD, source MAC ID
void TX_sendLpacket(Lpacket &lp); // send an Lpacket
void TX_sendTrailer(void); // send FCS, ED
BOOL TX_headerComplete; // header has been sent
BOOL TX_LpacketComplete; // Lpacket has been sent
BOOL TX_trailerComplete; // trailer has been sent
BOOL TX_abort; // DLPDU was aborted

/////////////////////////////////////////////////////////////////
//
// interface to LLC
//
Lpacket LLC_Lpacket; // shared variable between LLC and ACM containing the
Lpacket
BOOL LLC_pickLpacket (BOOL scheduled,
UINT octetsLeft); // tell LLC to pick an Lpacket for transmission
LLC LpacketSent(void); // tell LLC that the Lpacket was sent
/////////////////////////////////////////////////////////////////
//
// interface to station management
//

//
// various events that can be reported to SM
//
typedef enum
{
    DLL_EV_rxGoodFrame,
    DLL_EV_txGoodFrame,
    DLL_EV_badFrame,
    DLL_EV_errA,
    DLL_EV_errB,
    DLL_EV_txAbort,
    DLL_EV_NUT_overflow,
    DLL_EV_dribble,
    DLL_EV_nonconcurrency,
    DLL_EV_rxAbort,
    DLL_EV_lonely,
    DLL_EV_dupNode,
    DLL_EV_noisePulse,
    DLL_EV_collision,
    DLL_EV_invalidModAddress,
    DLL_EV_rogue,
    DLL_EV_deafness,
    DLL_EV_supernode,
}event;

extern void DLL_event(event ev, int opt=0); // report an event
extern void DLL_currentMod_indication(USINT macID);
extern void DLL_tminus_zero_indication(void);

extern void DLL_online_confirm(BOOL en);
extern void DLL_listen_only_confirm(BOOL en);
extern void DLL_tone_indication(USINT NUT_count);

class DLL_config_data
{
public:

    USINT myaddr;

    UINT NUT_length;
    USINT smax;
    USINT umax;
    USINT slotTime;
    USINT blanking;
    USINT gb_start;
    USINT gb_center;
    USINT interval_count;
    USINT modulus;
    USINT gb_prestart;
};

```

```

// interface to DLL management interface

extern DLL_config_data DLL_SM_pending;
extern DLL_config_data DLL_SM_current;

extern BOOL SM_powerup;

extern BOOL SM_mod_enable; // state of mod_enable
extern BOOL SM_online; // state of on-line/off-line
extern BOOL SM_online_req; // pending state of on-line/off-line
extern BOOL SM_listen_only; // listen only state
extern BOOL SM_listen_only_req; // pending listen only state

/////////////////////////////////////////////////////////////////
//
// subroutines
//
// take care of common processing for network parameter changes
//
void handle_net_change(void)
{
    if(SM_listen_only != SM_listen_only_req) // handle completion of listen_only
change
    {
        SM_listen_only = SM_listen_only_req;
        DLL_listen_only_confirm(SM_listen_only);
    }

    if(tMinus==1) // handle completion of synchronized change
    {
        tMinus = 0;
        DLL_tminus_zero_indication();
    }
}

//
// maintenance work done whenever the moderator is received
//
void processModerator(moderatorLpacket &moderator)
{
    //
    // copy data from the moderator
    // which cannot cause a rogue condition
    //
    interval_count = moderator.interval_count;
    usr = moderator.usr;
    tMinus = moderator.tMinus;

    lowcount = LOWCOUNTINIT; // re-initialize lowman counter

    // resynchronize the NUT timer to the moderator
    NUT_timer_count = DLL_SM_current.gb_center - MODERATOR_LENGTH/10;

    if(netState == MOD) // if this node is moderator
    {
        // and a moderator was received!
        netState = NOTMOD; // drop moderatorship
        modcount = 0; // init count of missing moderators
        DLL_event(DLL_EV_nonconcurrency); // indicate non-concurrency
    }
    else
    {
        if(!in_guardband) // unexpected moderator
        {
            DLL_event(DLL_EV_nonconcurrency); // indicate non-concurrency
        }

        if(netState == WATCH) // got moderator, there is a network
        {
            netState = DUPCHECK; // go check for dupnodes
            modcount = modcountinit; // this is how many NUTs to check
        }
        else if(netState == DUPCHECK) // if checking for duplicate MAC IDs
        {
            if(SM_listen_only == FALSE) // unless this node is required to
            {
                // remain in this state
                modcount--; // decrement NUT counter
                if(modcount == 0) // when done
                {
                    netState = NOTMOD; // it's OK to go transmit
                }
            }
        }
        else if(netState == ROGUE) // a rogue that sees a good moderator
    }
}

```

```

    {
        netState = DUPCHECK;          // can exit rogue state
        modcount = modcountinit;
    }
else
    {
        modcount = 0;                // else just reset mod counter
    }
}

//
// recovery after a time-out while waiting for moderator
//
void missed_moderator(void)
{
    if(netState == NOTMOD)            // if supposed to be on-line and hearing moderators
    {
        modcount++;                  // count consecutive missing moderator
        if(modcount >= TXNOMOD)      // if exceeded limit
        {
            netState = DUPCHECK;     // change to a quietly waiting mode
            modcount = 1;            // init mods required before return on-line
        }

        if(SM_listen_only == FALSE   // if allowed to transmit
           && SM_mod_enable == TRUE    // and allowed to be moderator
           && DLL_SM_current.myaddr <= nqlowman) // and apparently the lowman
        {
            lowcount--;              // count consecutive NUTs as lowman
            if(lowcount == 0)        // if this is the second time
            {
                netState = MOD;      // then activate the moderator on third NUT
            }
        }
    }
else
    {
        lowcount = LOWCOUNTINIT;  // otherwise, re-initialize lowman counter
    }
}

//
// housekeeping actions at the end of each NUT
//
void housekeeping(void)
{
    if(netState == WATCH)           // in watch state
    {
        lonely = LONELYINIT;        // hold the lonely counter reset
        if(watch_timer.count == 0)  // if timed out (no moderators)
        {
            // change to on-line -- try to build a link
            if (SM_listen_only == FALSE // if allowed to transmit
               && SM_mod_enable == TRUE // and allowed to moderate
               && nqlowman == DLL_SM_current.myaddr) // and lowman
            {
                netState = MOD;      // activate moderator state
            }
        }
    }
else
    {
        netState = DUPCHECK;        // else wait for another node to moderate
        modcount = modcountinit;
    }
}

else if (qlowman == 255             // no good FCS received in the last NUT
        && DLL_SM_current.myaddr != 0 // local node is not supernode
        && netState != WATCH)       // not already in the watch state
{
    lonely--;
    if(lonely == 0 || netState == ROGUE || netState == DUPNODE)
    {
        DLL_event(DLL_EV_lonely);
        if(netState == ROGUE)
        {
            watch_timer.start(0,75 sec);
        }
        else
        {
            watch_timer.start(1,25 sec);
        }
        netState = WATCH;
    }
}

```

```

    }
    else
    {
        lonely = LONELYINIT;
    }

    if(deafcount >= DEAFNESS)
    {
        deafcount = 0;
        DLL_event(DLL_EV_deafness);
        NUT_timer.count += DEAFADJUST;
    }

    // this counter determines how many NUTS a node should
    // check the link before deciding that it is not
    // a dupnode. if 0<myaddr<=SMAX a node is guaranteed to
    // an access each NUT, so the check time is low. On the
    // other hand, if smax<myaddr<=umax, a node might have to wait
    // a long time for its access slot come up, so use a long
    // time. If a DUPCHECKING unscheduled only node ever sees its slot
    // go by, it sets modcount to a lower number immediately.

    // nodes between SMAX and UMAX aren't guaranteed to get a slot every NUT
    // if not an unscheduled only node, use a low number else use a huge number

    if (DLL_SM_current.myaddr > DLL_SM_current.smax)
        modcountinit = TXDUPCHECKS;
    else
        modcountinit = 255;

    // update the interval count once per NUT
    interval_count = (interval_count + 1) % DLL_SM_current.modulus;

    // update the usr once per NUT
    usr = (usr + 1) % (DLL_SM_current.umax + 1);
}

/////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////
//
// The ACM State Machine
//
state: powerup

    event:          SM_powerup
    destination:    offline

//
// be idle until told to go on-line
//
state: offline

    // normal case
    event:          SM_online == TRUE
    condition:      ph_frame_indication == FALSE
    destination:    rxMod0
    action:
        in_guardband = TRUE; // start up at the beginning of a guardband
        if(DLL_SM_current.myaddr == 0) // either local node is supernode
        {
            netState = MOD;
            SM_listen_only = FALSE;
        }
        else // or not a supernode
        {
            netState = WATCH;
            watch_timer.start(1,25 sec);
        }
        scheduled = FALSE;
        in_guardband = TRUE;
        DLL_online_confirm(SM_online);

// exception: there was an Lpacket on the wire at transition to on-line
event:          SM_online == TRUE
condition:      ph_frame_indication == TRUE
destination:    badMod // treat it as an error inside the guardband
action:
    DLL_event(DLL_EV_badFrame,macSrce);
    in_guardband = TRUE;
    if(DLL_SM_current.myaddr == 0)
    {
        netState = MOD;
        SM_listen_only = FALSE;
    }
    else
    {
        netState = WATCH;
        watch_timer.start(1,25 sec);
    }

```

```

    }
    DLL_online_confirm(SM_online);

//
// wait for either energy on the wire or a slot time-out
//
state: waitFrame

    event:    ph_lock_indication == TRUE
    destination: frEst1
    action:    gen_timer.begin_counting();

    event:    slot_timer.expired()
    destination: gap
    action:    gen_timer.start(0,6 usec);

//
// wait for frame start, or noise
//
state: frEst1
    event:    ph_frame_indication == TRUE
    destination: rxFrame

    event:    ph_lock_indication == FALSE    // short energy burst == noise blip
    destination: waitFrame
    action:    DLL_event(DLL_EV_noisePulse);    // ignore it

    event:    gen_timer.count >= 8,0 usec
    destination: frEst2

//
// continue waiting
//
state: frEst2

    event:    ph_frame_indication == TRUE
    destination: rxFrame

    event:    ph_lock_indication == FALSE    // longer burst treat as a damaged DLPDU
    destination: endFrame
    action:    DLL_event(DLL_EV_badFrame,macSrc);
               gen_timer.start(5,2 usec);

    event:    gen_timer.count >= 11,2 usec    // energy this long with no data start
    destination: endFrame    // treat as bad DLPDU, no start delimiter
    action:    DLL_event(DLL_EV_badFrame,macSrc);
               gen_timer.start(5,2 usec);

//
// wait here for there to be no detectable energy on the wire (no clock lock)
//
state: endFrame

    event:    ph_lock_indication == FALSE
    destination: gap
    action:    slot_timer.restart();

//
// simulate first half of node turnaround time with this timing gap
//
state: gap

    event:    ph_frame_indication == TRUE    // DLPDU start overrides all
    destination: rxFrame

    event:    gen_timer.expired()    // process the rest of
    destination: nextSlot
    action:    gen_timer.start(6,1 usec);    // start second half of turnaround

    if(scheduled)    // do implicit token rotation, scheduled
    {
        itr++;
        if(itr>DLL_SM_current.smax)    // after SMAX,
        {
            // change from scheduled to unscheduled
            itr=usr;
            scheduled = FALSE;
        }
    }
    else    // implicit token rotation, unscheduled
    {

```



```

        itr = (itr + 1) % (DLL_SM_current.umax + 1);
    }

//
// decide what comes next
//
state: nextSlot

    event:    ph_frame_indication == TRUE        // detect a DLPDU start at all times
    destination: rxFrame

    // time for the guardband
    event:    gen_timer.expired()
    condition: NUT_timer.count < DLL_SM_current.gb_prestart
    destination: guardBand
    action:

        // scheduled time not completed -- log the error and proceed
        if(scheduled == TRUE)
        {
            DLL_event(DLL_EV_NUT_ouerrun);
            scheduled = FALSE;
        }
        in_guardband = TRUE;

    // not this node's slot, keep counting.
    // Note that the slot timer auto repeats to maintain regular slot intervals
    event:    gen_timer.expired()
    condition: NUT_timer.count >= DLL_SM_current.gb_prestart
                && itr != DLL_SM_current.myaddr
    destination: waitFrame

    // this node's slot, but not allowed to talk
    event:    gen_timer.expired()
    condition: NUT_timer.count >= DLL_SM_current.gb_prestart
                && itr == DLL_SM_current.myaddr
                && !(netState == MOD || netState == NOTMOD)
    destination: waitFrame
    action:    modcount = min(modcount, TXDUPCHECKS); // optimize dupcheck

    // this node's slot, and allowed to talk
    event:    gen_timer.expired()
    condition: NUT_timer.count >= DLL_SM_current.gb_prestart
                && itr == DLL_SM_current.myaddr
                && (netState == MOD || netState == NOTMOD)
    destination: waitForHold

//
// a DLPDU has started, this deals with the data
//
state: rxFrame

    // prematurely lost the DLPDU event
    event:    ph_frame_indication == FALSE
    destination: endFrame
    action:
        DLL_event(DLL_EV_badFrame, macSrce);
        gen_timer.start(5, 2 usec);

        // start hold timer after line is quiet
        holdoff_timer.start((DLL_SM_current.blanking+1) * 1, 6 usec);

    // duplicate node condition
    event:    RX_dataReady
    condition: RX_rxData == DLL_SM_current.myaddr
    destination: dupNode
    action:
        macSrce = RX_rxData;
        nqlowman = min(nqlowman, macSrce);

    // record the source MAC ID
    event:    RX_dataReady
    condition: RX_rxData != DLL_SM_current.myaddr
    destination: nextLpacket
    action:
        macSrce = RX_rxData;
        nqlowman = min(nqlowman, macSrce);

//
// start here to parse each new Lpacket
// the RxM delivers one complete Lpacket at a time
//
state: nextLpacket

    // received an out-of-sequence moderator -- attempt resync
    event:    RX_receivedLpacket
    condition: RX_Lpacket.service == MODERATOR_TAG
    destination: rxMod3 // unexpected moderator

```

```
// ignore most other received Lpackets
event:    RX_receivedLpacket
condition: RX_Lpacket.service != MODERATOR_TAG
destination: nextLpacket

// lost the DLPDU in the middle
event:    ph_frame_indication == FALSE
destination: endFrame
action:
    DLL_event(DLL_EV_badFrame,macSrce);
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// the DLPDU was aborted
event:    RX_abort
destination: endFrame
action:
    DLL_event(DLL_EV_rxAbsort);
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// if bad FCS, or DLPDU extended into the guardband, treat as bad DLPDU
event:    RX_endMAC
condition: NUT_timer.count <= DLL_SM_current.gb_start || !RX_FCSOK
destination: endFrame
action:
    DLL_event(DLL_EV_badFrame,macSrce);
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// good DLPDU, and not guardband time
// use the source ID
event:    RX_endMAC
condition: NUT_timer.count > DLL_SM_current.gb_start && RX_FCSOK
destination: endFrame
action:
    qlowman = min(qlowman, macSrce);
    DLL_event(DLL_EV_rxGoodFrame);
    if (itr != macSrce) DLL_event(DLL_EV_nonconcurrency);
    itr = macSrce;
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// deal with a duplicate node indication, local MAC ID already in by another node
//
state: dupNode

// bad DLPDU so ignore indication
event:    ph_frame_indication == FALSE
destination: endFrame
action:
    DLL_event(DLL_EV_badFrame,macSrce);
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// bad DLPDU so ignore indication
event:    RX_endMAC
condition: !RX_FCSOK
destination: endFrame
action:
    DLL_event(DLL_EV_badFrame,macSrce);
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// good DLPDU so accept the duplicate indication
event:    RX_endMAC
condition: RX_FCSOK
destination: endFrame
action:
    netState = DUPNODE;
    if (!dupflag)
    {
        dupflag == TRUE;
        DLL_event(DLL_EV_dupNode);
    }
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// Hold off transmission until blanking + 1 octet has expired
```

```

// since the last transmit or receive. It is possible that the
// hold timer expires before the gen_timer, dealing with turnaround time.
//
state: waitForHold

    // time to transmit, but line has activity, treat it as a collision, and back off.
    event: holdoff_timer.expired()
    condition: ph_lock_indication == TRUE
    destination: endFrame
    action:
        DLL_event(DLL_EV_collision);
        gen_timer.start(0,6 usec);

    // no problem, start transmission
    event: holdoff_timer.expired()
    condition: ph_lock_indication == FALSE
    destination: sendHeader
    action:
        // calculate time remaining in this NUT
        tmp = (unsigned)(NUT_timer.count-DLL_SM_current.gb_prestart);
        octetsleft = min(255, (tmp<1) + tmp + (tmp>>3)) * 2;
        TX_sendHeader(DLL_SM_current.myaddr);

//
// when header has been sent, start first Lpacket
//
state: sendHeader

    // TxLLC has something, send a new Lpacket
    event: TX_headerComplete
    condition: LLC_pickLpacket(scheduled, octetsleft) == TRUE
    destination: sendNextLpacket
    action:
        TX_sendLpacket(LLC_Lpacket);
        octetsleft = octetsleft - LLC_Lpacket.wire_size();

    // nothing to send that fits in the time left, close the DLPDU
    event: TX_headerComplete
    condition: LLC_pickLpacket(scheduled, octetsleft) == FALSE
    destination: sendTrailer
    action:
        TX_sendTrailer();
        if(scheduled // if unable to send all scheduled Lpackets
            && LLC_pickLpacket(scheduled, octetsleft) == TRUE)
        {
            DLL_event(DLL_EV_dribble);
        }

//
// send subsequent Lpackets
//
state: sendNextLpacket

    // TxLLC has something, send a new Lpacket
    event: TX_LpacketComplete
    condition: LLC_pickLpacket(scheduled, octetsleft) == TRUE
    destination: sendNextLpacket
    action:
        TX_sendLpacket(LLC_Lpacket);
        octetsleft = octetsleft - LLC_Lpacket.wire_size();

    // nothing to send that fits in the time left, close the DLPDU
    event: TX_LpacketComplete
    condition: LLC_pickLpacket(scheduled, octetsleft) == FALSE
    destination: sendTrailer
    action:
        TX_sendTrailer();

//
// complete DLPDU termination
//
state: sendTrailer

    event: TX_trailerComplete
    destination: endFrame
    action:
        if(TX_abort == TRUE) DLL_event(DLL_EV_txAbort);
        else DLL_event(DLL_EV_txGoodFrame);
        gen_timer.start(5,2 usec);
        // start hold timer after line is quiet
        holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// almost guardband start time, wait for guardband start then check lots of things
//
state: guardBand

    event: ph_frame_indication == TRUE // received DLPDU takes precedence
    destination: rxFrame // deal with it

```

```

// net change not in progress, this node is not moderator
event: NUT_timer.count <= DLL_SM_current.gb_start // guardband start time NOW
condition: tMinus == 0
          && SM_listen_only == SM_listen_only_req
          && netState != MOD
destination: rxMod0

// net change not in progress, this node is moderator AND can remain moderator
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: tMinus == 0
          && SM_listen_only == SM_listen_only_req
          && netState == MOD
          && SM_mod_enable == TRUE
          && min(DLL_SM_current.myaddr, qlowman) == DLL_SM_current.myaddr
destination: txMod0

// net change not in progress, this node is moderator, but cannot remain moderator
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: tMinus == 0
          && SM_listen_only == SM_listen_only_req
          && netState == MOD
          && (SM_mod_enable == FALSE
              || min(DLL_SM_current.myaddr, qlowman) != DLL_SM_current.myaddr)
destination: rxMod0
action:
    netState = NOTMOD;
    modcount = 0;
    lowcount = LOWCOUNTINIT;

// net change in progress, this node is not moderator
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: tMinus > 1 && SM_listen_only == SM_listen_only_req && netState != MOD
destination: rxMod0
action:
    tMinus = tMinus - 1;

// net change in progress, this node is moderator, and can remain moderator
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: tMinus > 1
          && SM_listen_only == SM_listen_only_req
          && netState == MOD
          && SM_mod_enable == TRUE
          && min(DLL_SM_current.myaddr, qlowman) == DLL_SM_current.myaddr
destination: txMod0
action:
    tMinus = tMinus - 1;

// net change in progress, this node is moderator, but cannot remain moderator
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: tMinus > 1
          && SM_listen_only == SM_listen_only_req
          && netState == MOD
          && (SM_mod_enable == FALSE
              || min(DLL_SM_current.myaddr, qlowman) != DLL_SM_current.myaddr)
destination: rxMod0
action:
    tMinus = tMinus - 1;
    netState = NOTMOD;
    modcount = 0;
    lowcount = LOWCOUNTINIT;

// net change complete, this node becomes supernode
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: tMinus == 1 && DLL_SM_pending.myaddr == 0
destination: txMod0
action:
    handle_net_change();
    netState = MOD;
    SM_listen_only = FALSE; // override listen_only

// net change complete, but shall stop transmitting for a time
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: ( tMinus == 1
            && DLL_SM_pending.myaddr != 0
            && DLL_SM_pending.myaddr != DLL_SM_current.myaddr
            || SM_listen_only != SM_listen_only_req
            && SM_listen_only)
destination: rxMod0
action:
    handle_net_change();
    netState = WATCH; // go to WATCH state
    watch_timer.start(1,25 sec);

// net change complete, this node is not moderator, no significant changes
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: ( tMinus == 1
            && DLL_SM_pending.myaddr != 0
            && DLL_SM_pending.myaddr == DLL_SM_current.myaddr

```

```

        || SM_listen_only != SM_listen_only_req
        && !SM_listen_only)
        && netState != MOD
destination: rxMod0
action:
    handle_net_change();

// net change complete, this node is moderator, no major changes, and can remain
moderator
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: (
    tMinus == 1
    && DLL_SM_pending.myaddr != 0
    && DLL_SM_pending.myaddr == DLL_SM_current.myaddr
    || SM_listen_only != SM_listen_only_req
    && !SM_listen_only)
    && netState == MOD
    && SM_mod_enable == TRUE
    && min(DLL_SM_pending.myaddr, qlowman) == DLL_SM_pending.myaddr
destination: txMod0
action:
    handle_net_change();

// net change complete, this node is moderator, no major changes, but can't stay
moderator
event: NUT_timer.count <= DLL_SM_current.gb_start
condition: (
    tMinus == 1
    && DLL_SM_pending.myaddr != 0
    && DLL_SM_pending.myaddr == DLL_SM_current.myaddr
    || SM_listen_only != SM_listen_only_req
    && !SM_listen_only)
    && netState == MOD
    && (SM_mod_enable == FALSE
    || min(DLL_SM_pending.myaddr, qlowman) != DLL_SM_pending.myaddr)
destination: rxMod0
action:
    handle_net_change();
    netState = NOTMOD;
    modcount = 0;
    lowcount = LOWCOUNTINIT;

//
// wait for moderator start
//
state: rxMod0

// got it
event: ph_frame_indication == TRUE
destination: rxMod1

// timed out waiting for moderator
event: NUT_timer.count <= 20 usec
destination: waitTone
action:
    missed_moderator();
    housekeeping();

//
// receive the moderator
//
state: rxMod1

// bad moderator
event: ph_frame_indication == FALSE
destination: badMod
action:
    DLL_event(DLL_EV_badFrame, macSrce);

// get MAC source ID
event: RX_dataReady == TRUE
destination: rxMod2
action:
    macSrce = RX_rxData;
    nqlowman = min(nqlowman, macSrce);

//
// continue receiving moderator
//
state: rxMod2

// bad DLPDU
event: ph_frame_indication == FALSE
destination: badMod
action:
    DLL_event(DLL_EV_badFrame, macSrce);

// bad moderator DLPDU if it's null
event: RX_endMAC
destination: badMod

```

```

action:
    DLL_event(DLL_EV_badFrame,macSrce);

// got an Lpacket, save the Lpacket for later, and go check the rest of the DLPDU
event:    RX_receivedLpacket
destination: rxMod3

//
// handle the end of a moderator DLPDU
//
state: rxMod3

// bad DLPDU
event:    ph_frame_indication == FALSE
destination: badMod
action:
    DLL_event(DLL_EV_badFrame,macSrce);

// bad DLPDU
event:    RX_abort
destination: badMod
action:
    DLL_event(DLL_EV_rxAbort);

// bad DLPDU if another Lpacket is in the moderator DLPDU
event:    RX_receivedLpacket
destination: badMod
action:
    DLL_event(DLL_EV_badFrame,macSrce);

// bad DLPDU
event:    RX_endMAC
condition: !RX_FCSOK
destination: badMod
action:
    DLL_event(DLL_EV_badFrame,macSrce);

// dupnode!
event:    RX_endMAC
condition: RX_FCSOK && macSrce == DLL_SM_current.myaddr
destination: endFrame
action:
    netState = DUPNODE;
    if(!dupflag)
    {
        dupflag = TRUE;
        DLL_event(DLL_EV_dupNode);
    }
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// good DLPDU, but bad header
event:    RX_endMAC
condition: RX_FCSOK
            && macSrce != DLL_SM_current.myaddr
            && ( RX_Lpacket.size != MODERATOR_SIZE
                || RX_Lpacket.ct1 != MODERATOR_CTL
                || RX_Lpacket.service != MODERATOR_TAG
                || RX_Lpacket.dest != 0xFF)
destination: badMod

// moderator, but from an incorrect address (not lowman)
event:    RX_endMAC
condition: RX_FCSOK
            && macSrce != DLL_SM_current.myaddr
            && RX_Lpacket.size == MODERATOR_SIZE
            && RX_Lpacket.ct1 == MODERATOR_CTL
            && RX_Lpacket.service == MODERATOR_TAG
            && RX_Lpacket.dest == 0xFF
            && macSrce > qlowman
destination: endFrame
action:
    DLL_event(DLL_EV_invalidModAddress);
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// this is a moderator from a supernode,
// or this node is at 0xFF so that all others look like a supernode
// adopt all the parameters from the moderator
event:    RX_endMAC
condition: RX_FCSOK
            && macSrce != DLL_SM_current.myaddr
            && RX_Lpacket.size == MODERATOR_SIZE
            && RX_Lpacket.ct1 == MODERATOR_CTL
            && RX_Lpacket.service == MODERATOR_TAG
            && RX_Lpacket.dest == 0xFF
            && macSrce <= qlowman

```

```

        && (macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
destination: waitTone
action:
    glowman = min(glowman, macSrce);
    if (macSrce == 0 && currentMod != 0) DLL_event(DLL_EV_supernode);
    currentMod = macSrce;
    DLL_currentMod_indication(currentMod);
    DLL_SM_current.NUT_length = moderator.NUT_length;
    DLL_SM_current.gb_prestart = moderator.gb_prestart;
    DLL_SM_current.gb_start = moderator.gb_start;
    DLL_SM_current.gb_center = moderator.gb_center;
    DLL_SM_current.modulus = moderator.modulus;
    DLL_SM_current.blanking = moderator.blanking;
    DLL_SM_current.slotTime = moderator.slotTime;
    DLL_SM_current.smax = moderator.smax;
    DLL_SM_current.umax = moderator.umax;
    processModerator(moderator);
    housekeeping();

// good moderator received, everything matches local copy
// resynchronize and continue
event:    RX_endMAC
condition: RX_FCSOK
    && macSrce != DLL_SM_current.myaddr
    && RX_Lpacket.size == MODERATOR_SIZE
    && RX_Lpacket.ctrl == MODERATOR_CTL
    && RX_Lpacket.service == MODERATOR_TAG
    && RX_Lpacket.dest == 0xFF
    && macSrce <= glowman
    && !(macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
    && moderator.NUT_length == DLL_SM_current.NUT_length
    && moderator.gb_prestart == DLL_SM_current.gb_prestart
    && moderator.gb_start == DLL_SM_current.gb_start
    && moderator.gb_center == DLL_SM_current.gb_center
    && moderator.modulus == DLL_SM_current.modulus
    && moderator.blanking == DLL_SM_current.blanking
    && moderator.slotTime == DLL_SM_current.slotTime
    && moderator.smax == DLL_SM_current.smax
    && moderator.umax == DLL_SM_current.umax
destination: waitTone
action:
    glowman = min(glowman, macSrce);
    currentMod = macSrce;
    DLL_currentMod_indication(currentMod);
    processModerator(moderator);
    housekeeping();

// The moderator doesn't match, so this node is a rogue.
// Since in the guardband, go to rxmod for recovery
event:    RX_endMAC
condition: RX_FCSOK
    && macSrce != DLL_SM_current.myaddr
    && RX_Lpacket.size == MODERATOR_SIZE
    && RX_Lpacket.ctrl == MODERATOR_CTL
    && RX_Lpacket.service == MODERATOR_TAG
    && RX_Lpacket.dest == 0xFF
    && macSrce <= glowman
    && !(macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
    && !(
        moderator.NUT_length == DLL_SM_current.NUT_length
        && moderator.gb_prestart == DLL_SM_current.gb_prestart
        && moderator.gb_start == DLL_SM_current.gb_start
        && moderator.gb_center == DLL_SM_current.gb_center
        && moderator.modulus == DLL_SM_current.modulus
        && moderator.blanking == DLL_SM_current.blanking
        && moderator.slotTime == DLL_SM_current.slotTime
        && moderator.smax == DLL_SM_current.smax
        && moderator.umax == DLL_SM_current.umax)
    && in_guardband == TRUE
destination: rxMod0
action:
    glowman = min(glowman, macSrce);
    currentMod = macSrce;
    DLL_currentMod_indication(currentMod);
    netState = ROGUE;
    DLL_event(DLL_EV_rogue);

// The moderator doesn't match, so this node is a rogue.
// Since currently not in the guardband, go to endframe for recovery.
event:    RX_endMAC
condition: RX_FCSOK
    && macSrce != DLL_SM_current.myaddr
    && RX_Lpacket.size == MODERATOR_SIZE
    && RX_Lpacket.ctrl == MODERATOR_CTL
    && RX_Lpacket.service == MODERATOR_TAG
    && RX_Lpacket.dest == 0xFF
    && macSrce <= glowman
    && !(macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
    && !(
        moderator.NUT_length == DLL_SM_current.NUT_length
        && moderator.gb_prestart == DLL_SM_current.gb_prestart

```



```

        && moderator.gb_start == DLL_SM_current.gb_start
        && moderator.gb_center == DLL_SM_current.gb_center
        && moderator.modulus == DLL_SM_current.modulus
        && moderator.blanking == DLL_SM_current.blanking
        && moderator.slotTime == DLL_SM_current.slotTime
        && moderator.smax == DLL_SM_current.smax
        && moderator.umax == DLL_SM_current.umax)
        && in_guardband == FALSE
destination: endFrame
action:
    qlowman = min(qlowman, macSrce);
    currentMod = macSrce;
    DLL_currentMod_indication(currentMod);
    netState = ROGUE;
    DLL_event(DLL_EV_rogue);
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

//
// come here to handle a bad moderator DLPDU
// wait for link to become quiet, or until tone generation is due
//
state: badMod

// This state is reached when this node identifies a moderator DLPDU,
// but the DLPDU is subsequently reported bad so the state transfers
// back to endFrame if this node is not in the guardband.
event: TRUE
condition: in_guardband == FALSE
destination: endFrame
action:
    DLL_event(DLL_EV_badFrame,macSrce);
    gen_timer.start(5,2 usec);
    // start hold timer after line is quiet
    holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

// the guardband should be ended NOW
event: NUT_timer.count <= 30 usec
destination: waitTone
action:
    missed_moderator();
    housekeeping();

// else wait until the link is quiet
event: ph_lock_indication == FALSE
destination: rxMod0

//
// send the moderator, wait for guardband center
//
state: txMod0

event: NUT_timer.count <= DLL_SM_current.gb_center
destination: txMod1
action:
    TX_sendHeader(DLL_SM_current.myaddr); // send header

//
// send more
//
state: txMod1

event: TX_headerComplete
destination: txMod2
action:
    moderator.size = MODERATOR_SIZE;
    moderator.ct1 = MODERATOR_CTL;
    moderator.service = MODERATOR_TAG;
    moderator.dest = 0xff;
    moderator.NUT_length = DLL_SM_current.NUT_length;
    moderator.smax = DLL_SM_current.smax;
    moderator.umax = DLL_SM_current.umax;
    moderator.slotTime = DLL_SM_current.slotTime;
    moderator.blanking = DLL_SM_current.blanking;
    moderator.gb_start = DLL_SM_current.gb_start;
    moderator.gb_center = DLL_SM_current.gb_center;
    moderator.usr = usr;
    moderator.interval_count = interval_count;
    moderator.modulus = DLL_SM_current.modulus;
    moderator.tMinus = tMinus;
    moderator.gb_prestart = DLL_SM_current.gb_prestart;
    moderator.spare = 0;

    TX_sendLpacket(moderator); // send the Lpacket

//
// finish up

```

```

//
state: txMod2

    event:      TX_LpacketComplete
    destination: txMod3
    action:
        TX_sendTrailer();

//
// at completion, transferring to waiting for tone
//
state: txMod3

    event:      TX_trailerComplete
    destination: waitTone
    action:
        currentMod = DLL_SM_current.myaddr;
        DLL_currentMod_indication(currentMod);
        housekeeping();

//
// wait for tone
//
state: waitTone

    // end of this NUT, and transferring to off-line
    event:      NUT_timer.count == 0
    condition:   SM_online == FALSE
    destination: offline
    action:
        DLL_tone_indication(interval_count);
        DLL_online_confirm(SM_online);

    // end of this NUT, start of another
    event:      NUT_timer.count == 0
    condition:   SM_online_req == TRUE
    destination: waitSlotZero
    action:
        DLL_tone_indication(interval_count);
        NUT_timer.restart();
        scheduled = TRUE;
        in_guardband = FALSE;

        // housekeeping

        // If this node detects line activity, but not moderators, attempt to recover.
        // Maybe problem is that the local node is performing housekeeping (and is
        // therefore deaf) during the time the moderator DLPDU is transmitted.

        if ((netState != MOD || netState != NOTMOD)
            && qlowman != 255
            && ph_frame_indication == TRUE)
        {
            deafcount++;
        }
        else
        {
            deafcount = 0;
        }

        gen_timer.start(20 usec); // start timer for start of scheduled time

//
// wait for start of scheduled
//
state: waitSlotZero

    // start a new scheduled token pass
    event:      gen_timer.expired()
    destination: gap
    action:
        itr = -1;
        if (DLL_SM_current.myaddr == 0) // supernode doesn't try to detect lowman
        {
            nqlowman = 0;
            qlowman = 0;
        }
        else // else initialize the detector
        {
            nqlowman = 255;
            qlowman = 255;
        }
        slot_timer.restart();
        gen_timer.start(0,6 usec);

```

9.3 TxLLC

The TxLLC (transmit LLC) shall receive and buffer Lpackets from the upper layers. It shall pick the next Lpacket to be transmitted based on

- the order in which Lpackets were queued;
- attributes of the Lpacket;
- information provided by the Access Control Machine (ACM).

The TxLLC shall present the selection Lpacket to the ACM for transmission.

```
// DLL TX LLC State Machine Description

// This state machine accepts transmit requests from the DLS-user,
// queues and prioritizes them, and submits one Lpacket at a time
// to the ACM based on parameters received from the ACM.

////////////////////////////////////

//
// type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;
typedef void *IDENTIFIER;

typedef enum { M_0,
               M_1,
               M_ND_plus,
               M_ND_minus } M_SYMBOL;

// These are the three transmit priorities.
// hard assignments are so that the priority can be
// used as an index into an array of FIFOs
// note that HIGH and LOW are unscheduled

typedef enum { SCHEDULED=0,
               HIGH=1,
               LOW=2 } PRIORITY;

typedef enum { OK,
               TXABORT,
               FLUSHED } TXSTATUS; // describes the result of a transmit request.

//
// Lpacket class
//
class Lpacket
{
public:

    // return the size octet of the current Lpacket
    USINT size;

    // return the ctl octet of the current Lpacket
    USINT ctl;

// Lpacket constants: masks for ctl octet

#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

    // return the value of the tag pad bit
    int tag_pad(void)
    {
        return (ctl&TAGPAD) >>1;
    }

    // return the value of the data pad bit
    int data_pad(void)
    {
        return (ctl&DATAPAD) >>2;
    }

    // return the number of octets in the Lpacket
    int wire_size(void)
    {
        return size*2 - tag_pad() - data_pad();
    }

    // store next octet to the Lpacket
    void put_octet(USINT data);

    // get an octet from the Lpacket
```

```

    USINT &operator[](int index);

    Lpacket(void *p)          // constructor
    {
    }

    Lpacket(int size)          // constructor
    {
    }
};

//
// superclass of Lpacket that defines an Lpacket to be transmitted
//
class txLpacket : public Lpacket
{
public:

    IDENTIFIER id;
    BOOL fixed;

    // constructors

    txLpacket(void *p) : Lpacket(p)
    {
    }

    txLpacket(int size) : Lpacket(size) // size is size of PDU buffer
    {
        // (Lpacket plus pads, in octets)
    }
};

/////////////////////////////////////////////////////////////////
//
// interface to DLS-user
//
void DLL_xmit_fixed_request(
    IDENTIFIER id,
    USINT      Lpacket[],
    UINT       size,
    PRIORITY   priority,
    USINT      service,
    USINT      destID);

extern void DLL_xmit_fixed_confirm (IDENTIFIER id, TXSTATUS status);

void DLL_xmit_generic_request(
    IDENTIFIER id,
    USINT      Lpacket[],
    UINT       size,
    PRIORITY   priority,
    USINT      tag[3]);

extern void DLL_xmit_generic_confirm (IDENTIFIER id, TXSTATUS status);

void DLL_flush_requests-by-QoS_request (PRIORITY priority);
extern void DLL_flush_requests-by-QoS_confirm (PRIORITY priority);
DLL_flush_single_request( PRIORITY priority, IDENTIFIER xmit_id );

/////////////////////////////////////////////////////////////////
//
// interface to ACM
//
txLpacket *pickLpacket(BOOL scheduled, int octetsLeft); // pick an Lpacket to send next
void LpacketSent(TXSTATUS status);                     // the Lpacket was sent

/////////////////////////////////////////////////////////////////
//
// interface to station management
//
void SM_powerup(void); // input indication: powerup has occurred

/////////////////////////////////////////////////////////////////
//
// a class to represent message FIFOs
//
class FIFO
{
public:

    void put(txLpacket lp); // add an Lpacket to the FIFO
    txLpacket &get(void);   // remove an Lpacket from the FIFO
    txLpacket &peek(void);  // look at the first Lpacket in the FIFO
};

```

```

void flush(void);           // delete all Lpackets in the FIFO
void flush1(IDENTIFIER id); // delete one Lpacket in the FIFO, given it's ID
BOOL has_data();           // true if the FIFO is not empty
};

FIFO fifo[3];               // at least three priority levels shall be provided

////////////////////////////////////
//
// TxLLC implementation
//
//
// powerup initialization
//
void SM_powerup(void)
{
    fifo[SCHEDULED].flush();
    fifo[HIGH].flush();
    fifo[LOW].flush();
}

//
// flush a particular FIFO
//
void DLL_flush-requests-by-QoS_request (PRIORITY priority)
{
    fifo[priority].flush();
    DLL_flush-requests-by-QoS_confirm(priority);
}

//
// flush a particular Lpacket
//
void DLL_flush_single_request (PRIORITY priority, IDENTIFIER id)
{
    fifo[priority].flush1(id);
}

//
// requests from DLS-user to submit Lpackets for transmission
//
void DLL_xmit_fixed_request(
    IDENTIFIER id,
    USINT      Lpacket[],
    UINT       size,
    PRIORITY   priority,
    USINT      service,
    USINT      destID)
{
    int tmp;

    tmp = size + 4 + (size&1); // size of DLSdu plus 4 octet fixed Lpacket header
                                // plus optional data pad

    txLpacket &lp(new txLpacket(tmp)); // create a new Lpacket buffer

    // build the PDU (Lpacket)
    lp[0] = tmp/2; // size, in words
    lp[1] = 0x01 | ((size&1)<<2); // ctl octet, plus data pad bit, if needed
    lp[2] = service; // fixed tag service
    lp[3] = destID; // destination MAC ID
    memcpy(&lp[4], Lpacket, size); // copy the rest of the data
    // there may be a pad octet sent after the DLSdu
    lp.id = id; // store the user's id (for the confirmation)
    lp.fixed = TRUE; // store type of Lpacket (fixed)
    fifo[priority].put(lp); // queue the Lpacket
}

void DLL_xmit_generic_request(
    IDENTIFIER id,
    USINT      Lpacket[],
    UINT       size,
    PRIORITY   priority,
    USINT      tag[3])
{
    int tmp;
    int pad;

    tmp = size + 6 + (size&1); // octet size of DLSdu plus 6 octet GEN Lpacket header
                                // plus data pad
    pad = 0;

    txLpacket &lp(new txLpacket(tmp)); // create a new Lpacket buffer

    // build the PDU (Lpacket)
    lp[0] = tmp/2; // size, in words

```

```

    lp[1] = 0x12 | ((size&1)<<2);    // control octet,
                                   // plus maybe a data pad bit,

    lp[2] = 0;                      // tag pad octet (affects memory image of Lpacket)
    lp[3] = tag[0];                  // generic tag
    lp[4] = tag[1];                  // generic tag
    lp[5] = tag[2];                  // generic tag
    memcpy(&lp[6], Lpacket, size);   // copy the rest of the data
                                   // there may be pad octet after the DLSdu

    lp.id = id;                      // store the user's identifier (for the confirm)
    lp.fixed = FALSE;                // store type of Lpacket (generic)

    fifo[priority].put(lp);          // queue the new Lpacket
}

static txLpacket *picked = 0;        // remember which Lpacket was picked

//
// called by ACM to pick the next Lpacket to be sent out
//
txLpacket *pickLpacket(BOOL scheduled, int octetsLeft)
{
    int wordsleft = (octetsLeft+1)/2; // wordsleft is rounded up, accepting that in
    some                               // cases this results in an Lpacket not being sent

    if(scheduled)                    // if scheduled, only look at scheduled FIFO
    {
        // if there is anything in the FIFO and it fits in the time left
        if(fifo[SCHEDULED].has_data()
        && fifo[SCHEDULED].peek().size <= wordsleft)
        {
            picked = &fifo[SCHEDULED].get(); // then pick it to be sent
        }
        else picked = 0;              // no Lpacket available
    }
    else                             // if unscheduled -- look at all FIFOs in order
    {
        // if there is anything in the FIFO and it fits in the time left
        if(fifo[SCHEDULED].has_data()
        && fifo[SCHEDULED].peek().size <= wordsleft)
        {
            picked = &fifo[SCHEDULED].get(); // then pick it to be sent
        }
        else if(fifo[HIGH].has_data() // ditto for HIGH
        && fifo[HIGH].peek().size <= wordsleft)
        {
            picked = &fifo[HIGH].get();
        }
        else if(fifo[LOW].has_data()  // ditto for LOW
        && fifo[LOW].peek().size <= wordsleft)
        {
            picked = &fifo[LOW].get();
        }
        else picked = 0;              // no Lpacket available
    }

    return picked;
}

//
// confirmation from the ACM that the picked Lpacket was sent (or possibly aborted)
//
void LpacketSent(TXSTATUS status)
{
    if(picked->fixed)                 // if the Lpacket was fixed
    {
        DLL_xmit_fixed_confirm (picked->id, status); // generate a fixed confirm
    }
    else                             // else
    {
        DLL_xmit_generic_confirm (picked->id, status); // generate a generic confirm
    }

    delete picked;                  // delete temp data
    picked = 0;
}

```

9.4 RxLLC

The RxLLC (receive LLC) shall buffer Lpackets from the Receive Machine (RxM) as they are assembled. When the RxM indicates that a good DLPDU has concluded, all buffered Lpackets shall be presented to the upper layers in the order received. If the RxM indicates a bad DLPDU has concluded, all buffered Lpackets shall be discarded.

```
// DLL RX LLC State Machine Description

// This state machine gets Lpackets from the RxM,
// and DLPDU status from the deserializer.
// It handles quarantining, and passes quarantined
// Lpackets up to the DLS-user

/////////////////////////////////////////////////////////////////
//
// generic type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;

//
// Lpacket class
//
class Lpacket
{
public:

    // the size octet of the current Lpacket
    USINT size;

    // the ctl octet of the current Lpacket
    USINT ctl;

// Lpacket constants: masks for ctl octet

#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

    // return the value of the tag pad bit
    int tag_pad(void)
    {
        return (ctl&TAGPAD) >>1;
    }

    // return the value of the data pad bit
    int data_pad(void)
    {
        return (ctl&DATAPAD) >>2;
    }

    // return the number of octets in the Lpacket
    int wire_size(void)
    {
        return size*2 - tag_pad() - data_pad();
    }

    // store next octet to the Lpacket
    void put_octet(USINT data);

    USINT &operator[](int index);

    // init for a new Lpacket
    void init(void);

    Lpacket(void *p) {} // constructor
};

class rxLpacket : public Lpacket
{
public:
    USINT source; // the MAC ID of the node that sent this Lpacket
    int memory_size; // size of the Lpacket in memory
    BOOL fixed; // true if fixed , false if generic

    rxLpacket(void *p) : Lpacket(p) {}
};
```



```

/////////////////////////////////////////////////////////////////
//
// a class to represent message FIFOs
//
class FIFO
{
public:

void put(rxLpacket lp);          // add an Lpacket to the fifo
rxLpacket &get(void);           // remove an Lpacket from the fifo
void flush(void);               // delete all Lpackets in the FIFO
BOOL has_data();                // true if the fifo is not empty
};

/////////////////////////////////////////////////////////////////
//
// a class to represent generic tags
//
class gen_tag
{
USINT data[3];
public:
// constructor
gen_tag(USINT first, USINT second, USINT third)
{
data[0] = first;
data[1] = second;
data[2] = third;
}

USINT &operator[](int index)
{
return data[index];
}
};

/////////////////////////////////////////////////////////////////
//
// internal variables
//

FIFO fifo;

/////////////////////////////////////////////////////////////////
//
// interface to Phil
//
BOOL ph_lock_indication; // optimistic DLPDU detect (clock recovery is tracking)
BOOL ph_frame_indication; // pessimistic DLPDU detect (Manchester valid since the SD)

/////////////////////////////////////////////////////////////////
//
// interface to Deserializer
//
extern BOOL RX_endMAC; // the end delimiter has been detected
extern BOOL RX_FCSOK; // FCS on last DLPDU was OK
extern BOOL RX_abort; // abort received on last DLPDU

/////////////////////////////////////////////////////////////////
//
// interface to RxM
//
BOOL RX_receivedLpacket; // indication: an Lpacket has been received
rxLpacket RX_Lpacket(0); // data: the Lpacket most recently received

/////////////////////////////////////////////////////////////////
//
// Interface to DLS-user
//
DLL_rcv_fixed_indication (
USINT Lpacket[],
UINT size,
USINT service,
USINT sourceID);

DLL_rcv_generic_indication (
USINT Lpacket[],
UINT size,
gen_tag tag);

```

```

////////////////////////////////////
//
// interface to station management
//
BOOL SM_powerup;    // input indication: powerup has occurred

////////////////////////////////////
//
// RxLLC states
//

// wait for powerup

state: powerup

    event:    SM_powerup
    destination: idle
    action:
        fifo.flush();

// wait for a DLPDU to start
state: idle

    event:    ph_frame_indication == TRUE
    destination: getLpacket

state: getLpacket

    // stuff a new Lpacket into FIFO
    event:    RX_receivedLpacket    // wait for a new Lpacket to be assembled
    destination: getLpacket
    action:
        fifo.put(RX_Lpacket);    // stuff the Lpacket into the FIFO

    // if FCS is good, give all Lpackets to next layer up
    event:    RX_endMAC
    condition: RX_FCSOK
    destination: idle
    action:
        rxLpacket &lp(0);
        while(fifo.has_data())
        {
            lp = fifo.get();
            if(lp.fixed)
            {
                DLL_recv_fixed_indication(
                    &lp[4],    // rip off 4 octets of header to leave DLSdu
                    lp.memory_size-4,
                    lp[2],    // send service octet
                    lp.source);    // send source MAC ID
            }
            else
            {
                DLL_recv_generic_indication(
                    &lp[6],    // rip off 6 octets of header to leave DLSdu
                    lp.memory_size-6,
                    gen_tag(lp[3],lp[4],lp[5]));    // send a three octet tag
            }
        }

    // if DLPDU is damaged, flush all the Lpackets in the FIFO
    event:    RX_endMAC
    condition: !RX_FCSOK
    destination: idle
    action:
        fifo.flush();    // discard all Lpackets

```

9.5 Transmit machine (TxM)

The transmit machine (TxM) shall receive commands and data from the Access Control Machine to transmit the components of a DLPDU. It shall transmit a DLPDU by passing data octets and commands to the Serializer.

```
// DLL TxM State Machine Description

// This state machine gets commands and Lpackets from the ACM to
// build and transmit a DLPDU. It uses the services of the
// Serializer to do this.

/////////////////////////////////////////////////////////////////
//
// generic type and constant definitions
//

typedef enum {FALSE=0, TRUE=1} BOOL;

typedef void *IDENTIFIER;

/////////////////////////////////////////////////////////////////
//
// Lpacket definition
//
// Lpacket constants: masks for ctl octet
//
#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

//
// Lpacket class
//
class Lpacket
{
public:

    USINT    size;
    USINT    ctl;

    // return the value of the tag pad bit
    int tag_pad(void)
    {
        return (ctl&TAGPAD) >>1;
    }

    // return the value of the data pad bit
    int data_pad(void)
    {
        return (ctl&DATAPAD) >>2;
    }

    // return the number of octets in the Lpacket
    int wire_size(void)
    {
        return size*2 - tag_pad() - data_pad();
    }

    // get the next octet to be transmitted
    USINT get_next_octet(void);

    // is this Lpacket aborted?
    BOOL abort(void);
};

/////////////////////////////////////////////////////////////////
//
// internal variables
//
int counter;

/////////////////////////////////////////////////////////////////
//
// interface from ACM and TxLLC
//
BOOL TX_sendHeader;           // command: send preamble, SD, source MAC ID
USINT myaddr;                 // input: the source MAC ID
BOOL TX_headerComplete;       // reply: header has been sent

BOOL TX_sendLpacket;          // command: send an Lpacket
class Lpacket lp;              // input: the Lpacket to be sent
BOOL TX_LpacketComplete;      // reply: Lpacket has been sent
```

```

BOOL TX_sendTrailer;          // command: send FCS, ED
BOOL TX_done;                 // reply: DLPDU is complete

BOOL TX_abort;                // status: DLPDU was aborted

////////////////////////////////////
//
// interface to Serializer
//
void SER_txRequest(BOOL state); // command: turn transmitter on and off
void SER_sendData(USINT data); // command: send a Manchester coded data octet
void SER_sendSD(void);         // command: send a start delimiter
void SER_sendED(void);         // command: send an end delimiter
void SER_clearFCS(void);       // command: clear the FCS accumulator
void SER_sendFCS(void);        // command: send the FCS

BOOL SER_operationComplete;    // reply: current operation is completed

////////////////////////////////////
//
// interface to station management
//
BOOL SM_powerup;               // event indication

////////////////////////////////////
//
// TxM states
//
// wait for powerup
//
state: powerup

    event: SM_powerup
    destination: idle

//
// wait for a DLPDU to send
//
state: idle

    // when commanded by ACM to start a DLPDU, send the first preamble
    event: TX_sendHeader
    destination: pre0
    action:
        TX_done = FALSE;
        TX_headerComplete = FALSE;
        TX_abort = FALSE;
        SER_txRequest(TRUE); // turn on transmitter
        SER_sendData(0xff); // send first preamble octet

//
// send second preamble octet
//
state: pre0

    event: SER_operationComplete
    destination: pre1
    action:
        SER_sendData(0xff); // send second preamble octet

//
// send start delimiter
//
state: pre1

    event: SER_operationComplete
    destination: sd
    action:
        SER_sendSD(); // send source MAC ID

//
// send MAC ID
//
state: sd

    event: SER_operationComplete
    destination: finishHeader
    action:
        SER_clearFCS();
        SER_sendData(myaddr);

//
// wait for header complete
//

```

```

state: finishHeader

    event:    SER_operationComplete
    destination: nextLP
    action:
        TX_headerComplete = TRUE;

//
// wait for a command to either send an Lpacket or to terminate the DLPDU
//
state: nextLP

    // start a new Lpacket, grab the size octet
    event:    TX_sendLpacket
    destination: sendCtl
    action:
        counter = lp.wire_size() - 1; // note size zero is not a permitted value
        SER_sendData(lp.get_next_octet());
        TX_LpacketComplete = FALSE;

    // terminate DLPDU: send the FCS
    event:    TX_sendTrailer
    destination: sendFCS
    action:
        SER_sendFCS();

state: sendCtl

    // sent the CTL octet, ignoring tag pad if present
    event:    SER_operationComplete
    destination: sendData
    action:
        counter = counter - 1;
        SER_sendData(lp.get_next_octet());
        if (lp.tag_pad()) // if tag pad
        {
            lp.get_next_octet(); // discard next octet
        }

//
// send the rest of the data in an Lpacket
//
state: sendData

    // if aborted, send an abort sequence: SD followed immediately by ED
    event:    SER_operationComplete
    condition: lp.abort()
    destination: abort1
    action:
        SER_sendData(0xff); // required before the Start Delimiter to avoid
        // violating run length requirement of phy layer

    // if more data, send the next octet
    event:    SER_operationComplete
    condition: !lp.abort() && counter > 0
    destination: sendData
    action:
        counter = counter - 1;
        SER_sendData(lp.get_next_octet());

    // when done, wait for the next command
    event:    SER_operationComplete
    condition: !lp.abort() && counter <= 0
    destination: nextLP
    action:
        if (lp.data_pad()) // if data pad
        {
            lp.get_next_octet(); // discard next octet
        }
        TX_LpacketComplete = TRUE; // signal that Lpacket is done

//
// send the ED
//
state: sendFCS

    event:    SER_operationComplete
    destination: sendED
    action:
        SER_sendED();

//
// wait for ED complete, then shut down and wait for next DLPDU
//
state: sendED

    event:    SER_operationComplete
    destination: idle

```

```

    action:
        SER_txRequest(FALSE);    // turn off transmitter
        TX_done = TRUE;          // signal that DLPDU is done

//
// sent an FF before the abort, when it's done send an Start Delimiter
//
state: abort1

    event:    SER_operationComplete
    destination: abort2
    action:
        SER_sendSD();

//
// send second part of the abort (End Delimiter)
//
state: abort2

    event:    SER_operationComplete
    destination: abort3
    action:
        SER_sendED();

//
// wait for abort complete, then shut down and wait for next DLPDU
//
state: abort3

    event:    SER_operationComplete
    destination: idle
    action:
        SER_txRequest(FALSE);    // turn off transmitter
        TX_abort = TRUE;         // assert error flag
        TX_LpacketComplete = TRUE; // signal that Lpacket is done
        TX_done = TRUE;          // signal that DLPDU is done

```

9.6 Receive machine (RxM)

The receive machine (RxM) shall receive framing information and data octets from the Deserializer. It shall decode Lpackets from the received octet stream for presentation to the ACM and the RxLLC. It shall accept generic and fixed Lpackets for which the corresponding tag filter is enabled, and discard Lpackets that are not enabled.

```

// DLL RxM State Machine Description

// This state machine gets octet symbols from the Deserializer and uses
// them to build Lpackets, which are then sent to the RxLLC.

////////////////////////////////////
//
// generic type and constant definitions
//

typedef enum {FALSE=0, TRUE=1} BOOL;

typedef void *IDENTIFIER;

////////////////////////////////////
//
// Lpacket definition
//
// Lpacket constants: masks for ctl octet
//
#define FIXEDSCREEN 1
#define TAGPAD 2
#define DATAPAD 4

//
// Lpacket class
//
class Lpacket
{
public:

    // the size octet of the current Lpacket
    USINT size;

    // the ctl octet of the current Lpacket
    USINT ctl;

    // return the value of the tag pad bit

```

[illegible]


```
// internal variables
//
// the tag filtering tables
class gen_screener gen;
class fixed_screener fixed;

// MAC ID of source of DLPDU
USINT source;

// an octet counter
int counter;

// temps for generic screener tag octets
USINT gen0;
USINT gen1;

////////////////////////////////////
//
// interface to PhL
//
extern BOOL ph_lock_indication; // optimistic DLPDU detect (clock recovery is
tracking)
extern BOOL ph_frame_indication; // pessimistic DLPDU detect (Manchester valid since
the SD)

////////////////////////////////////
//
// interface to Deserializer
//
extern BOOL RX_dataReady; // an octet of a DLPDU is available
extern USINT RX_rxData; // octet most recently received from DLPDU (source
MAC ID)
extern BOOL RX_endMAC; // the end delimiter has been detected
extern BOOL RX_FCSOK; // FCS on last DLPDU was OK
extern BOOL RX_abort; // abort received on last DLPDU

////////////////////////////////////
//
// interface to station management
class DLL_config_data
{
public:
USINT myaddr;
UINT NUT_length;
USINT smax;
USINT umax;
USINT slotTime;
USINT blanking;
USINT gb_start;
USINT gb_center;
USINT interval_count;
USINT modulus;
USINT gb_prestart;
};

extern DLL_config_data SM_current; // the DLL config variables -- need my MAC ID
extern BOOL SM_powerup; // input indication: powerup has occurred

////////////////////////////////////
//
// interface to RxM
//

// received data interface

BOOL RX_receivedLpacket; // an Lpacket has been received
rxLpacket RX_Lpacket; // the Lpacket most recently received

// interface to manage tag filters

extern enable_generic_confirm (IDENTIFIER id, BOOL result);
extern disable_generic_confirm (IDENTIFIER id, BOOL result);
extern enable_fixed_confirm (IDENTIFIER id, BOOL result);
extern disable_fixed_confirm (IDENTIFIER id, BOOL result);

void DLL_enable_generic_request (IDENTIFIER id, gen_tag tag)
{
enable_generic_confirm (id, gen.enable(tag));
}

void DLL_disable_generic_request (IDENTIFIER id, gen_tag tag)
```

```

    {
        disable_generic_confirm (id, gen.disable(tag));
    }

void DLL_enable_fixed_request (IDENTIFIER id, USINT service)
{
    enable_fixed_confirm (id, fixed.enable(service));
}

void DLL_disable_fixed_request (IDENTIFIER id, USINT service)
{
    disable_fixed_confirm (id, fixed.disable(service));
}

////////////////////////////////////
//
//    RxM states
//
//
//
// wait for powerup
//
state: powerup

    event:    SM_powerup
    destination: idle
    action:
        RX_receivedLpacket = FALSE;

//
// wait for a DLPDU to start
//
state: idle

    event:    ph_frame_indication == TRUE
    destination: getID

state: getID

    // if DLPDU is terminated, wait for next DLPDU
    event:    ph_frame_indication == FALSE
    destination: idle

    // get the source MAC ID and save it for all the Lpackets in the DLPDU
    event:    RX_dataReady == TRUE
    destination: getSize
    action:
        source = RX_rxData; // save the source MAC ID for subsequent Lpackets

state: getSize

    // if DLPDU is terminated, wait for next DLPDU
    event:    ph_frame_indication == FALSE
    destination: idle

    // get the size octet and init a new Lpacket
    event:    RX_dataReady == TRUE
    destination: getCtl
    action:
        RX_receivedLpacket = FALSE; // re-initialize the interface
        RX_Lpacket.init();
        RX_Lpacket.source = source; // save the source in the new Lpacket
        RX_Lpacket.put_octet(RX_rxData); // store the size octet

state: getCtl

    // if DLPDU is terminated, wait for next DLPDU
    event:    ph_frame_indication == FALSE
    destination: idle

    // get the CTL octet for a fixed Lpacket
    event:    RX_dataReady == TRUE && (RX_rxData & 0xFB) == 0x01 // fixed tag
    destination: getFixed0
    action:
        RX_Lpacket.put_octet(RX_rxData); // store the ctl octet
        counter = RX_Lpacket.wire_size() - 2; // init the octet counter

    // get the CTL octet for a generic Lpacket
    event:    RX_dataReady == TRUE && (RX_rxData & 0xFB) == 0x12 // generic tag
    destination: getGen0
    action:
        RX_Lpacket.put_octet(RX_rxData); // store the ctl octet
        RX_Lpacket.put_octet(0); // store the pad octet
        counter = RX_Lpacket.wire_size() - 2; // init the octet counter

```

```
// get the CTL octet for a generic Lpacket
event:    RX_dataReady == TRUE
          && (RX_rxData & 0xFB) != 0x01
          && (RX_rxData & 0xFB) != 0x12
destination: ignoreRest
action:
    counter = RX_Lpacket.wire_size() - 2; // init the octet counter

//
// handle service octet of fixed Lpacket
//
state: getFixed0

    // if DLPDU is terminated, wait for next DLPDU
    event:    ph_frame_indication == FALSE
    destination: idle

    // handle case if tag is enabled
    event:    RX_dataReady == TRUE
              && fixed.lookup(RX_rxData)
    destination: getFixed1
    action:
        RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
        counter--; // count it

    // handle case if tag is disabled
    event:    RX_dataReady == TRUE
              && !fixed.lookup(RX_rxData)
    destination: ignoreRest
    action:
        counter--; // count it

//
// handle dest octet of fixed Lpacket
//
state: getFixed1

    // if DLPDU is terminated, wait for next DLPDU
    event:    ph_frame_indication == FALSE
    destination: idle

    // handle case if addressed to me or broadcast
    event:    RX_dataReady == TRUE
              && (RX_rxData == SM_current.myaddr || RX_rxData == 0xFF)
    destination: getRest
    action:
        RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
        counter--; // count it
        RX_Lpacket.fixed = TRUE;

    // handle case if not addressed to me
    event:    RX_dataReady == TRUE
              && RX_rxData != SM_current.myaddr
              && RX_rxData != 0xFF
    destination: ignoreRest
    action:
        counter--; // count it

//
// handle first tag octet of generic Lpacket
//
state: getGen0

    // if DLPDU is terminated, wait for next DLPDU
    event:    ph_frame_indication == FALSE
    destination: idle

    // save the next octet
    event:    RX_dataReady == TRUE
    destination: getGen1
    action:
        RX_Lpacket.put_octet(gen0=RX_rxData); // save and put the next octet into
Lpacket
        counter--; // count it

//
// handle second tag octet of generic Lpacket
//
state: getGen1

    // if DLPDU is terminated, wait for next DLPDU
    event:    ph_frame_indication == FALSE
    destination: idle

    // save the next octet
    event:    RX_dataReady == TRUE
```

```

    destination: getGen2
    action:
        RX_Lpacket.put_octet(gen1=RX_rxData); // save and put next data octet into
Lpacket
        counter--; // count it

//
// handle third octet of generic Lpacket
//
state: getGen2

    // if DLPDU is terminated, wait for next DLPDU
    event: ph_frame_indication == FALSE
    destination: idle

    // handle case if local node is interested in this generic Lpacket
    event: RX_dataReady == TRUE
            && gen.lookup(gen_tag(gen0, gen1, RX_rxData))
    destination: getRest
    action:
        RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
        counter--; // count it
        RX_Lpacket.fixed = FALSE;

    // handle case if not interested in this generic Lpacket
    event: RX_dataReady == TRUE
            && !gen.lookup(gen_tag(gen0, gen1, RX_rxData))
    destination: ignoreRest
    action:
        counter--; // count it

state: getRest

    // if DLPDU is terminated, wait for next DLPDU
    event: ph_frame_indication == FALSE
    destination: idle

    // get data octets except the last
    event: RX_dataReady == TRUE
    condition: counter > 1
    destination: getRest
    action:
        RX_Lpacket.put_octet(RX_rxData); // put the next data octet into the Lpacket
        counter--; // count it

    // get the last data octet; handle data pad, indicate arrived Lpacket; wait for
next Lpacket
    event: RX_dataReady == TRUE
    condition: counter == 1
    destination: getSize
    action:
        RX_Lpacket.put_octet(RX_rxData); // get the last octet
        if(RX_Lpacket.data_pad()) // if a data pad is requested,
        {
            RX_Lpacket.put_octet(0); // insert a zero
        }

        // remove pad octets from data count
        RX_Lpacket.memory_size -= RX_Lpacket.size*2;

        RX_receivedLpacket = TRUE; // notify that that an Lpacket has arrived

state: ignoreRest

    // if DLPDU is terminated, wait for next DLPDU
    event: ph_frame_indication == FALSE
    destination: idle

    // get data octets except the last
    event: RX_dataReady == TRUE
    condition: counter > 1
    destination: ignoreRest
    action:
        counter--; // count it

    // handle the last data octet; wait for next Lpacket
    event: RX_dataReady == TRUE
    condition: counter == 1
    destination: getSize

```

9.7 Serializer

For each command from the TxM, the Serializer shall produce the appropriate M_symbol stream, which shall be output to the medium via the PhL.

```
// DLL octet Serializer

// This state machine gets commands to transmit octet symbols from the TxM.
// It decomposes the octet symbols into M_symbols, and
// uses the services of the PHY layer to send these on the medium.
// This specifies the correct order of transmission of the various octet symbols.

// The FCS algorithm is the same as that used by HDLC

/////////////////////////////////////////////////////////////////
//
// data types
//
typedef enum {FALSE, TRUE} BOOL;

/////////////////////////////////////////////////////////////////
//
// interface to TxM
//

// requests (TxM to SER)

void SER_txRequest(BOOL state);      // command: turn transmitter on and off
void SER_sendData(USINT data);      // command: send a Manchester coded data octet
void SER_sendSD(void);              // command: send a start delimiter
void SER_sendED(void);              // command: send an end delimiter
void SER_clearFCS(void);             // command: clear the FCS accumulator
void SER_sendFCS(void);              // command: send the FCS

// indications (SER to TxM)

extern void SER_operationComplete(void); // notify the TxM that an operation is complete

/////////////////////////////////////////////////////////////////
//
// interface to the PHY layer
//
typedef enum {M_0, M_1, M_ND_plus, M_ND_minus} M_SYMBOL;

BOOL ph_frame_request; // transmit enable/disable
M_SYMBOL ph_data_request; // data to be sent

/////////////////////////////////////////////////////////////////
//
// internal variables
//
static unsigned int accum; // the FCS accumulator, 16 bits, sign is irrelevant
// as long as zeros shift in from the right

/////////////////////////////////////////////////////////////////
//
// internal subroutines
//

// add one octet to the FCS accumulator
// this is just one possible implementation of the HDLC FCS

static void FCS_octet(unsigned int d)
{
    accum ^= d & 0xff;

    for(int i=0;i<8;i++)
    {
        accum = (accum>>1) ^ (0x8408 * (accum&1)); // use FCS-CCITT polynomial
    }
}

/////////////////////////////////////////////////////////////////
//
// Serializer implementation
//

// requests (TxM to SER)
```

```

// enable transmitter
void SER_txRequest(BOOL state)
{
    ph_frame_request = state;
}

// add a data octet to the FCS and send it
void SER_sendData(USINT data)
{
    FCS_octet(data);

    // transmit data bits 0 through 7

    ph_data_request = (data & 0x01) ? M_1 : M_0;
    ph_data_request = (data & 0x02) ? M_1 : M_0;
    ph_data_request = (data & 0x04) ? M_1 : M_0;
    ph_data_request = (data & 0x08) ? M_1 : M_0;
    ph_data_request = (data & 0x10) ? M_1 : M_0;
    ph_data_request = (data & 0x20) ? M_1 : M_0;
    ph_data_request = (data & 0x40) ? M_1 : M_0;
    ph_data_request = (data & 0x80) ? M_1 : M_0;

    SER_operationComplete();    // tell TxM that operation is complete
}

// send a start delimiter
void SER_sendSD(void)
{
    ph_data_request = M_ND_plus;
    ph_data_request = M_0;
    ph_data_request = M_ND_minus;
    ph_data_request = M_ND_plus;
    ph_data_request = M_ND_minus;
    ph_data_request = M_1;
    ph_data_request = M_0;
    ph_data_request = M_1;
    SER_operationComplete();    // tell TxM that operation is complete
}

// send an end delimiter
void SER_sendED(void)
{
    ph_data_request = M_1;
    ph_data_request = M_0;
    ph_data_request = M_0;
    ph_data_request = M_1;
    ph_data_request = M_ND_plus;
    ph_data_request = M_ND_minus;
    ph_data_request = M_ND_plus;
    ph_data_request = M_ND_minus;
    SER_operationComplete();    // tell TxM that operation is complete
}

// initialize the FCS accumulator
void SER_clearFCS(void)
{
    accum = 0xffff;    // init the accumulator to all ones
    SER_operationComplete();    // tell TxM that operation is complete
}

// send the FCS
void SER_sendFCS(void)
{
    int tmp;

    tmp = accum ^ 0xffff;    // invert the FCS before sending;
    SER_sendData(tmp);    // do not calculate FCS while sending itself
    SER_sendData(tmp>>8);
    SER_operationComplete();    // tell TxM that operation is complete
}

```

9.8 Deserializer

9.8.1 Octet construction

The Deserializer shall accept $M\_symbols$ from $ph\_data\_indication$. $M\_symbols$ at the start of a frame shall be ignored until the $ph\_frame\_indication$ line transitions from false to true. This transition shall start on the first data octet – the source MAC ID. The Deserializer shall then group subsequent sets of eight MAC data symbols into octets. As each subsequent data octet is ready, the Deserializer shall transition the $RX\_dataReady$ line from false to true.

9.8.2 FCS checking

A modified CCITT Cyclic Redundancy Check shall be performed on the transferred data to verify the frame check sequence (FCS).

The formal concepts for the FCS generator and checker are described in 5.3.8.2.

The method used for cheking shall be the same as specified for HDLC (ISO/IEC 3309), with the exception that start and end delimiters shall be substituted for flags, and Manchester encoding shall be substituted for bit-stuffing. The FCS checker shall implement the polynomial $X^{16} + X^{12} + X^5 + 1$, and shall result in a two octets frame check sequence (FCS).

NOTE The last two data octets in the DLPDU (FCS octets) are calculated by the transmitting node so that, in the absence of errors, the remainder in the receiving node is always 0xF0B8 (see 5.3.8.2).

The receive FCS process shall begin by pre-setting the FCS checker to 0xFFFF when the `ph_frame_indication` line transitions from false to true. All subsequent data bits on the `ph_data_indication` line, excluding the end delimiter, shall be applied to the FCS checker. At end of receiving a DLPDU, the remainder (FCS) shall be compared to 0xF0B8.

9.8.3 End of DLPDU processing

Data octet processing shall proceed until `ph_frame_indication` transitions from true to false. This transition can happen mid-octet, but no action shall be taken until the start of the next octet. This time shall mark the end of reception for this frame.

The `ph_status_indication` received after `ph_frame_indication` transitions from true to false shall determine which of the following actions to take:

- if Normal – set `RX_FCSOK` to true if (FCS remainder == 0xF0B8); otherwise, set to false;
- if Abort – set `Rx_abort` to true; set `Rx_FCSOK` to false;
- if Invalid – set `Rx_abort` to false; set `Rx_FCSOK` to false.

9.9 DLL management

DLL Management shall buffer the station management variables that are required for the DLL to operate. It shall manage synchronized changes on these variables via interfaces to Station Management and the ACM.

```
// DLL Station Management Interface Description

// This component holds the station management variables that are
// subject to synchronized change.

////////////////////////////////////
//
// type and constant definitions
//
typedef enum {FALSE=0, TRUE=1} BOOL;

typedef void *IDENTIFIER;    // The data type of the identifier is unspecified.

// config data definition

class DLL_config_data
{
public:
    USINT  myaddr;
    UINT   NUT_length;
    USINT  smax;
    USINT  umax;
    USINT  slotTime;
    USINT  blanking;
    USINT  gb_start;
    USINT  gb_center;
    USINT  interval_count;
```



```

USINT modulus;
USINT gb_prestart;
};

/////////////////////////////////////////////////////////////////
//
// interface to station management
//
void DLL_tMinus-start-countdown-request (IDENTIFIER id, USINT start_count);
extern void DLL_tMinus-start-countdown-confirm (IDENTIFIER id, BOOL result);

void DLL_set_pending_request (IDENTIFIER id, DLL_config_data params);
extern void DLL_set_pending_confirm (IDENTIFIER id, BOOL result);

void DLL_get_pending_request (IDENTIFIER id);
extern void DLL_get_pending_confirm (IDENTIFIER id, DLL_config_data params);

void DLL_set_current_request (IDENTIFIER id, DLL_config_data params);
extern void DLL_set_current_confirm (IDENTIFIER id, BOOL result);

void DLL_get_current_request (IDENTIFIER id);
extern void DLL_get_current_confirm (IDENTIFIER id, DLL_config_data params);

/////////////////////////////////////////////////////////////////
//
// interface to ACM
//
extern void DLL_tminus_zero_indication (); // ACM calls this when tMinus=0 occurs
DLL_config_data DLL_SM_pending; // the ACM needs to see both current and pending
DLL_config_data DLL_SM_current;
extern void DLL_tminus_zero_indication (); // poke the ACM's tMinus counter to get
// a synchronized change started. The ACM
// continually writes over this every time the
// moderator is received, unless it is the moderator

void SM_supernode(void); // if this node saw a moderator sent by a supernode,
// any pending SM update shall be cancelled

/////////////////////////////////////////////////////////////////
//
// internal variables
//
BOOL pending_changed = FALSE;

/////////////////////////////////////////////////////////////////
//
// write the pending copy of DLL_config
void DLL_set_pending_request (IDENTIFIER id, DLL_config_data params)
{
    DLL_SM_pending = params; // save the params in the pending buffer
    pending_changed = TRUE;
    DLL_set_pending_confirm (id, TRUE);
}

// write the current copy of DLL_config
// ONLY USED WHEN OFFLINE
void DLL_set_current_request (IDENTIFIER id, DLL_config_data params)
{
    DLL_SM_current = params; // save the params in the pending buffer
    DLL_set_current_confirm (id, TRUE);
}

// read the pending copy of DLL_config
void DLL_get_pending_request (IDENTIFIER id)
{
    if (pending_changed == TRUE) // if pending has been updated
    {
        // return the contents of the pending buffer
        DLL_get_pending_confirm (id, DLL_SM_pending);
    }
    else // if pending buffer is invalid
    {
        // return the contents of the current buffer
        DLL_get_current_confirm (id, DLL_SM_current);
    }
}

//
// read the current copy of DLL_config
//
void DLL_get_current_request (IDENTIFIER id)
{
    // return the contents of the current buffer

```

```

    DLL_get_current_confirm (id , DLL_SM_current);
}

//
// called by the ACM when a tMinus countdown has completed
//
void DLL_tminus_zero_indication(void)
{
    if (pending_changed)
    {
        DLL_SM_current = DLL_SM_pending;
        pending_changed = FALSE;
    }
}

//
// start a synchronized change sequence
//
void DLL_tMinus_start_countdown_request (IDENTIFIER id, USINT start_count)
{
    ACM_tMinus = start_count;    // the ACM polls this variable
    DLL_tMinus_start_countdown_confirm (id, TRUE);
}

//
// cancel pending change if a supernode has been seen
//
void SM_supernode(void)
{
    pending_changed = FALSE;
}

```

Withdrawing

IECNORM.COM: Click to view the full PDF of IEC 61158-4-2:2007

Annex A

(normative)

Indicators and switches

A.1 Purpose

A common approach to status indicators and switch behavior is recommended to provide a consistent human user interface for all devices conforming to this standard. These indicators, typically LEDs, assist maintenance and diagnosis of problems with the Media, Physical and Data Link Layers.

NOTE Throughout this annex the term “network” is used to refer to that portion of the Ph-subnetwork and DL-subnetwork that is connected to the device making the observation.

A.2 Indicators

A.2.1 General indicator requirements

CPF 2 profiles have different levels of requirements for support of status indicators. However, if a device does support status indicators, they shall follow the specifications described in this annex for the selected CP.

NOTE 1 Indicators help maintenance personnel to quickly identify a faulty unit or media (e.g. red indicators are typically used to indicate a fault condition). For this reason, it is strongly recommended to implement these indicators in any device for which no practical constraint prevents it.

Two types of status indicators may be provided:

- one module status indicator;
- one or several network status indicators.

NOTE 2 These may be sometimes combined.

Additional indicators may be present; however, the naming and symbol conventions of the standard indicators shall not be employed for other indicators.

Safety devices have additional requirements, see IEC 61784-3-2.

A.2.2 Common indicator requirements

A.2.2.1 Applicability of common requirements

The common indicator requirement shall only apply to indicators for which requirements are specified in this standard.

A.2.2.2 Visibility of indicators

Indicators shall be visible without removing covers or parts from the equipment. Indicators shall be easily seen in normal lighting. Any related labels and icons shall be visible whether or not the indicator is illuminated.

A.2.2.3 Indicator flash rate

Unless otherwise indicated, a two state indicator shall have a flash rate of $(1,0 \pm 0,1)$ Hz. The indicator shall have a duty cycle of $(50 \pm 5)\%$.

NOTE The indicator is in the first state (on) for approximately 0,5 s and in the second state (off) for approximately 0,5 s.

A.2.2.4 Module status indicator

A.2.2.4.1 Description

This single bicolor (green/red) indicator shall provide status of the entire device. It shall indicate whether or not the device is powered, configured, and operating properly.

NOTE A device with more than one communication port would have only one module status indicator.

A.2.2.4.2 States

The module status indicator states shall be as specified in Table A.1. The indicator states reflect the device states specified in the Identity object (see IEC 61158-5-2).

Table A.1 – Module status indicator

Indicator state	Device state	Cause
Steady off	No power	There is no power applied to the device.
Steady green	Operational	The device is operating in a normal condition.
Flashing green	Standby	The device needs commissioning due to configuration missing, incomplete or incorrect.
Flashing red	Recoverable fault	The device has detected a recoverable major fault (see NOTE 1).
Steady red	Unrecoverable fault	The device has an unrecoverable major fault; may need replacing.
Flashing red / green	Device self testing	The device is executing its power up self tests.
NOTE 1 Indicator flash rates are given in A.2.2.3.		
NOTE 2 An incorrect or inconsistent configuration would be considered a recoverable fault.		

Any indicator colors/states not defined in Table A.1 are reserved and shall not be used.

Safety devices have additional requirements, see IEC 61784-3-2.

A.2.3 Fieldbus specific indicator requirements (1)

A.2.3.1 General requirements (1)

Two categories of status indicators shall be provided for each CP 2/1 device:

- one module status indicator;
- two network status indicators.

A.2.3.2 Indicators at power up

During power up, the indicators shall turn on to a fault or reset state. All indicators shall remain on (illuminated) for at least 1 s. After completion of power up, the indicator shall return to a normal operational state.

A.2.3.3 Module status indicator

Behavior of the module status indicator shall be as specified in A.2.2.4.

The module status indicator shall be labeled with one of the following:

- “MS”;
- “OK”;

“status”;

“mod status”;

“module status”.

A.2.3.4 Network status indicators

A.2.3.4.1 Description

The indication of network status shall require two bicolor (red/green) indicators. If the device has redundant PhLs (channels), each of the indicators shall be associated with one of the channels. If the device does not support channel redundancy, one of the indicators shall be associated with the single channel; however, two indicators shall still be required.

NOTE These indicators signify a number of conditions within the PhL; and when viewed together, they can reflect the state of the Data Link Layer.

The network status indicators do not usually reflect any errors received on the NAP; however, nodes which have only a NAP (no other PhL variants) may use the network status indicators to represent the status of the NAP. If this optional behavior is implemented, the state of both the indicators shall reflect the state of NAP communication, instead of the A and B channels.

A.2.3.4.2 States

A.2.3.4.2.1 Definitions

The following definitions shall be used to describe the indicator behavior:

steady – indicator shall be illuminated continuously;

alternating – both indicators, viewed together, shall alternate between two defined states, and the two indicators shall always be in opposite states, out of phase;

flashing – each indicator, viewed independently, shall alternate between two states, and if the indicators are flashing, they shall flash together, in phase.

A.2.3.4.2.2 Priority

The different states of the network status indicators shall follow a strict priority scheme. If more than one condition exists, the network status indicators shall display the state with the highest priority as specified in Table A.2.

Table A.2 – Network status indicators

Priority	Indicator state	How to view	Cause
highest (1)	both steady off	viewed together	reset or no power
2	both steady red		failed link interface
3	alternating red / green		self test
4	alternating red / off		bad node configuration (such as duplicate MAC ID)
5	steady off	viewed independently	channel disabled or channel not supported
6	flashing red / green		invalid link configuration
7	flashing red / off		link fault or no DLPDUs received
8	flashing green / off		temporary channel error, or listen-only
lowest (9)	steady green		normal operation

A.2.3.4.2.3 Steady green

During normal operation, when DLPDUs are being received without detected errors, the network status indicator for any enabled channel shall be steady green.

A.2.3.4.2.4 Steady red

If the link interface has faulted, both indicators shall be steady red.

A.2.3.4.2.5 Steady off

If one of the PhL entities is disabled, the indicator for the disabled channel shall be off. A channel can be disabled either because the device has only a single PhL or because it has two PhLs but exists on a non-redundant link. Only deactivating the entire network interface shall cause both indicators to be extinguished (off).

A.2.3.4.2.6 Flashing green / off

A flashing green / off pattern shall signify a temporary channel error. The two channels shall be treated independently with respect to these errors. An error on channel A shall be indicated by any of three events:

Ph_status_indication from channel A does not equal Normal;

DLL_badFCS_indication(CHA) is asserted;

DLL_event_indication(DLL_EV_errA) is asserted.

Likewise, an error on channel B shall be indicated by any of three events:

Ph_status_indication from channel B does not equal Normal;

DLL_badFCS_indication(CHB) is asserted;

DLL_event_indication(DLL_EV_errB) is asserted.

If more than one error occurs in a single DLPDU, only one channel error shall be declared. There shall be at most one error event per DLPDU per channel. Each PhL shall have an independent Ph_status_indication. The Data Link Layer shall report errors on at least the channel selected for presentation to higher levels. The DLL may also report errors on the other channel.

The flashing green state shall be entered whenever a channel error is detected. The flashing green state shall transition to the steady green state if no errors are detected for $4\text{ s} \pm 1\text{ s}$ on a given channel.

When the Data Link Layer is in a listen-only state, the indicator for any enabled channels shall flash green.

NOTE The ControlNet object can be used to force the Data Link Layer to enter the listen-only state.

A.2.3.4.2.7 Flashing red / off

A flashing red / off pattern shall signify a severe link error. If the Ph_frame_indication from one of channels remains false for a $1,0\text{ s} \pm 0,1\text{ s}$ period, the respective indicator shall assume the flashing red state for the next $1,0\text{ s} \pm 0,1\text{ s}$ period.

For devices with redundant channels enabled, if the error rate for one of the channels exceeds that of the other channel by more than one error per N DLPDUs received, then the channel with the higher rate shall flash red. N shall be in the range 10^6 through 2×10^6 . The interval used to measure channel error rate shall be 5×10^6 DLPDUs.

A.2.3.4.2.8 Flashing red / green

A flashing red / green pattern shall signify a bad link configuration. The network status indicators shall enter the flashing red / green state if the network attachment monitor (NAM) indicates that the node has been asked to use invalid or unsupported link parameters.

To optionally indicate a receive queue overflow or a transmit queue underflow, the network status indicators shall enter the flashing red / green state. The network status indicators shall remain in this state at least 3 s even if the overflow condition has cleared.

A.2.3.4.2.9 Alternating red / off

An alternating red / off pattern shall signify an invalid node configuration. This state shall be entered if the `DLL_event_indication` has either of the following values:

`DLL_EV_dupNode;`

`DLL_EV_rogue.`

This state shall be left immediately when the error condition clears.

A.2.3.4.2.10 Alternating red / green

An alternating red / green pattern shall signify that the Data Link Layer is in a self-test mode. The network status indicators shall remain in this state for at least 3 s even if the self-test has completed.

A.2.3.4.3 Labeling

The network status indicators shall be labeled using the symbols drawn in Figure A.1 and Figure A.2. The first channel shall be labeled with the outline of a channel icon. The second channel shall be labeled with a filled in channel icon. The first and second channels may be labeled with an uppercase A and B, respectively.

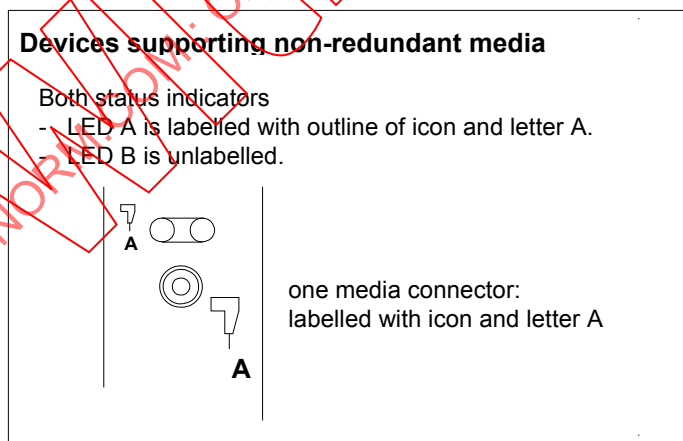


Figure A.1 – Non redundant network status indicator labeling

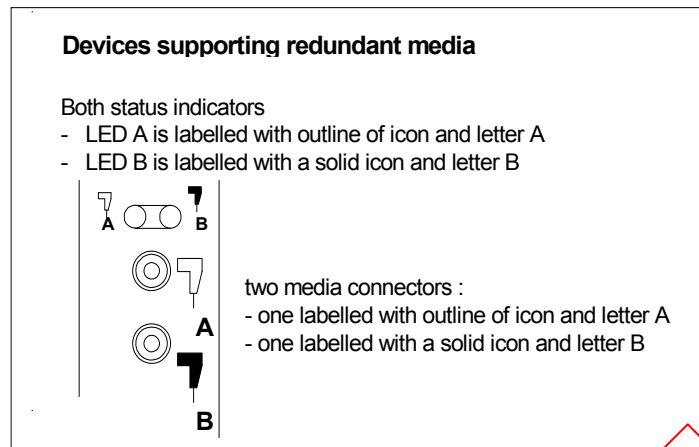


Figure A.2 – Redundant network status indicator labeling

A.2.4 Fieldbus specific indicator requirements (2)

A.2.4.1 General requirements (2)

CP 2/2 does not require a device to have indicators.

Two types of status indicators may be provided:

- one module status indicator;
- one or several network status indicators.

NOTE 1 The fieldbus supporting consortia (see <www.odva.org>) defines an Industrial Performance Level that requires a device to support both the module status and network status indicators as specified in this standard.

NOTE 2 A device with more than one communication port would have only one module status indicator, but more than one network status indicator (one per port).

NOTE 3 Devices are encouraged to have an indicator that displays the state of link (e.g. link status, tx/rx, collision) following generally accepted industry practices (as used in devices such as switches).

A.2.4.2 Indicators at power up

An indicator test shall be performed at power-up. To allow a visual inspection, the following sequence shall be performed:

- first indicator green, all other indicators off;
- first indicator on green for approximately 0,25 s;
- first indicator on red for approximately 0,25 s;
- first indicator on green;
- second indicator (if present) on green for approximately 0,25 s;
- second indicator (if present) on red for approximately 0,25 s;
- second indicator (if present) off.

If other indicators are present, each indicator shall be tested in sequence as prescribed by the second indicator above. If a Module Status indicator is present, it shall be the first indicator in the sequence, followed by any Network Status indicator present. After completion of this power up test, the indicator(s) shall turn to a normal operational state.

A.2.4.3 Module status indicator

Behavior of the module status indicator shall be as specified in A.2.2.4.

The module status indicator shall be labeled with one of the following:

- “MS”;
- “Mod”;
- “Mod status”;
- “Module status”.

A.2.4.4 Network status indicators

A.2.4.4.1 Description

The indication of network status shall require a single bicolor (red/green) indicator that represents the state of a single communication port.

NOTE A device with more than one communication port would have one network status indicator per port.

A.2.4.4.2 States

The network status indicator states shall be as specified in Table A.3.

Table A.3 – Network status indicator

Indicator state	Summary	Cause
Steady off	Not powered, no IP address	If the device does not have an IP address (or is powered off), the network status indicator shall be steady off
Flashing green	No connections	If the device has no established connections, but has obtained an IP address, the network status indicator shall be flashing green
Steady green	Connected	If the device has at least one established connection (even to the Message Router), the network status indicator shall be steady green
Flashing red	Connection timeout	If one or more of the connections in which this device is the target has timed out, the network status indicator shall be flashing red. This shall be left only if all timed out connections are reestablished or if the device is reset
Steady red	Duplicate IP	If the device has detected that its IP address is already in use, the network status indicator shall be steady red
Flashing green / red	Self-test	While the device is performing its power up testing, the network status indicator shall be flashing green / red

A.2.4.4.3 Labeling

The network status indicator shall be labeled with one of the following:

- “NS”;
- “Net”;
- “Net Status”;
- “Network Status”.

A.2.5 Fieldbus specific indicator requirements (3)

A.2.5.1 General requirements (3)

CP 2/3 does not require a device to have indicators.

If a device supports indicators, then it is recommended to provide:

- one module status indicator;

— one status indicators;
or a combined module/network status indicator.

A device may also support I/O status indicators.

If there is a conflict between turning an indicator on red versus green, the indicator shall be turned on red.

A.2.5.2 Indicators at power up

A.2.5.2.1 Module and network status indicators at power up

An indicator test shall be performed at power-up. To allow a visual inspection, the following sequence shall be performed:

- network status indicator off;
- module status indicator on green for approximately 0,25 s;
- module status indicator on red for approximately 0,25 s;
- module status indicator on green;
- network status indicator on green for approximately 0,25 s;
- network status indicator on red for approximately 0,25 s;
- network status indicator off.

If a device has other indicators each indicator should be tested in sequence.

A.2.5.2.2 Combined module/network status indicators at power up

An indicator test shall be performed at power-up. To allow a visual inspection, the following sequence shall be performed:

- combined module/network status indicator on green for approximately 0,25 s;
- combined module/network status indicator on red for approximately 0,25 s;
- combined module/network status indicator off.

If a device has other indicators each indicator should be tested in sequence.

A.2.5.3 Module status indicator

Behavior of the module status indicator shall be as specified in A.2.2.4.

The module status indicator should be labeled with one of the following:

- “MS”;
- “Module status”.

A.2.5.4 Network status indicator

A.2.5.4.1 Description

This bicolor (green/red) indicator provides the status of the communication link.

A.2.5.4.2 States

The network status indicator states shall be as specified in Table A.4. See the Link Access State Transition Diagram in IEC 62026-3 to compare the network status indicator to the Network Access State machine.

Table A.4 – Network status indicator

Indicator state	Summary	Cause
Steady off	Not powered/ not on-line	Device is not on-line. – The device has not completed the Dup_MAC_ID test yet. – The device may not be powered, look at module status indicator. – No network power present
Flashing green ^a	On-line, not connected	Device is on-line but has no connections in the established state. – The device has passed the Dup_MAC_ID test, is on-line, but has no established connections to other nodes. – For a Group 2 Only device it means that this device is not allocated to a master. – For a UCMM capable device it means that the device has no established connections
Steady green	Link OK, on-line, connected	The device is on-line and has connections in the established state. – For a Group 2 Only device it means that the device is allocated to a Master. – For a UCMM capable device it means that the device has one or more established connections
Flashing red ^a	Connection time-out	One or more I/O Connections are in the Timed-Out state
Steady red	Critical link failure	Failed communication device. The device has detected an error that has rendered it incapable of communicating on the network (duplicate MAC ID, or Bus-off)
Flashing red / green ^b	Communication faulted and received an Identify Comm Fault request - long protocol	A specific Communication Faulted device. The device has detected a Network Access error and is in the Communication Faulted state. The device has subsequently received and accepted an Identify Communication Faulted request - long protocol message
^a Indicator flash rates are given in A.2.2.3.		
^b Indicator flash rates are given in IEC 62026-3.		

Any indicator colors/states not defined in Table A.4 are reserved and shall not be used.

Safety devices have additional requirements, see IEC 61784-3-2.

A.2.5.4.3 Labeling

The network status indicator should be labeled with one of the following:

“NS”;

“Network Status”.

A.2.5.5 Combined module/network status indicator

A.2.5.5.1 Description

This bicolor (green/red) indicator provides limited device and communication status. The combined module/network status indicator indicates whether or not the device has power and is operating properly.

A.2.5.5.2 States

The combined module/network status indicator states shall be as specified in Table A.5. See the Link Access State Transition Diagram in IEC 62026-3 to compare the combined module/network status indicator to the Network Access State machine.

Table A.5 – Combined module/network status indicator

Indicator state	Summary	Cause
Steady off	Device not powered/ not on-line	Device is not on-line. – The device has not completed the Dup_MAC_ID test yet. – The device may not be powered
Steady green	Device operational and on-line, connected	The device is operating in a normal condition and the device is on-line with connections in the established state. – For a Group 2 Only device it means that the device is allocated to a Master. – For a UCMM capable device it means that the device has one or more established connections
Flashing green ^a	Device operational and on-line, not connected or Device on-line and device needs commissioning	The device is operating in a normal condition and the device is on-line with no connections in the established state. – The device has passed the Dup_MAC_ID test, is on-line, but has no established connections to other nodes. – For a Group 2 Only device it means that this device is not allocated to a master. – For a UCMM capable device it means that the device has no established connections. – Configuration missing, incomplete or incorrect
Flashing red ^a	Minor Fault and/or connection time-out and/or no network power	Any one or more of the following conditions. – Recoverable fault. – One or more I/O connections are in the Timed-Out state. – No network power present
Steady red	Critical fault or critical link failure	The device has an unrecoverable fault; may need replacing. Failed communication device. The device has detected an error that has rendered it incapable of communicating on the network (duplicate MAC ID, or Bus-off)
Flashing red / green ^b	Communication faulted and received an identify comm fault request - long protocol	A specific Communication Faulted device. The device has detected a Network Access error and is in the Communication Faulted state. The device has subsequently received and accepted an Identify Communication Faulted request - long protocol message
^a Indicator flash rates are given in A.2.2.3.		
^b Indicator flash rates are given in IEC 62026-3.		

Any indicator colors/states not defined in Table A.5 are reserved and shall not be used.

A combined module/network status indicator shall not be used for safety devices.

A.2.5.5.3 Labeling

The combined module/network status indicator should be labeled with one of the following:

- “MNS”;
- “Mod/Net Status”;
- “Module/Network Status”.

A.2.5.6 I/O status indicator

A.2.5.6.1 Description

This bicolor (green/red) indicator provides information concerning the states of inputs and/or outputs.

The intent of the I/O Status indicator is to inform the user whether this device has outputs under control and whether any outputs or inputs are active (e.g. outputs active, inputs producing) or faulted. The indicator is intended to reflect the mode/state of the inputs and outputs, not necessarily the on/off condition of the I/O points themselves.

A.2.5.6.2 States

The specific definition of the I/O status indicator colors and states reside in the individual definitions of the corresponding application objects that specify the use of this indicator.

Table A.6 provide guidelines governing the use of the I/O status indicator.

Table A.6 – I/O status indicator

Indicator state	Summary	Cause
Steady off	Output(s) inactive Input(s) inactive	All outputs are inactive. All inputs are inactive
Steady green	Output(s) active Input(s) active	One or more outputs are active and under control, and no outputs are “faulted”. One or more inputs are active and producing data, and no inputs are “faulted”
Flashing green	Output(s) idle	One or more outputs are idle and no outputs are active nor “faulted”
Flashing red	Output(s) faulted Inputs(s) faulted	One or more outputs are “faulted”, maybe in the fault state. One or more inputs are “faulted”, maybe in the fault state
Steady red	Output(s) forced off Input unrecoverable fault	One or more outputs are forced off (may be an unrecoverable Fault). One or more inputs has an unrecoverable fault
NOTE Indicator flash rates are given in A.2.2.3.		

Any indicator colors/states not defined in Table A.6 are reserved and shall not be used.

A.2.5.6.3 Labeling

The I/O status indicator should be labeled with one of the following:

- “IO”;
- “I/O”;
- “I/O status”.

A.3 Switches

A.3.1 Common switch requirements

A.3.1.1 General

All switches shall behave in the same way for the same function. For example, all network address switches for devices shall behave in the same way.

If a device has switches, then it is recommended that the device provides the means by which the switch setting can be determined remotely using messaging services.

Safety devices have additional requirements, see IEC 61784-3-2.

A.3.1.2 Software override of switches

If a user accessible switch (e.g. MAC ID or bit rate switch) can be overwritten by software, then an indicator shall be present that indicates whether or not it has been overwritten.

If an attribute is configurable with a switch and it cannot be overwritten with software, then the device shall respond with the error "Attribute not settable" to a service which attempts to set the attribute.

A.3.1.3 Times that switches are valid

Switches shall comply with either of the following rules:

- the switch shall be live at all times;
- the switch shall be live only at power turn on (PTO).

A.3.2 Fieldbus specific switch requirements (1)

A.3.2.1 Network address switches

A.3.2.1.1 Appearance

If the MAC ID is able to be set using switches, they shall be indicated in decimal format. This may be accomplished by using a rotary, thumbwheel, pushwheel, or other type of switch. The most significant digit shall be to the left or to the top of the least significant digit.

A.3.2.1.2 When to read switch

Network address switches shall only be read at power turn on (PTO). If the network address switches are changed after power turn on, a minor fault shall be detected and the module status indicator shall indicate a non-critical fault (flashing red). The MAC ID of the node used by the Data Link Layer shall remain at the value read during PTO. The current setting of the network address switches shall be stored in the ControlNet object so that another node may remotely diagnose the non-critical fault.

A.3.2.1.3 Labeling

The network address switches shall be labeled with one of the following:

- "NA";
- "address";
- "net address".

A.3.3 Fieldbus specific switch requirements (2)

There are no switch requirements for CP 2/2.

A.3.4 Fieldbus specific switch requirements (3)

A.3.4.1 Network address switches

A.3.4.1.1 Appearance

If DIP switches are used to set the MAC ID they shall be in binary format. If rotary, thumbwheel or pushwheel switches are used, then decimal format is required. The most significant digit shall always be to the left or to the top of the device when the user is trying to configure the switches.

A.3.4.1.2 Labeling

The network address switches shall be labeled with one of the following:

“NA”;

“Node Address”.

A.3.4.2 Bit rate switches

A.3.4.2.1 Appearance

If switches are used to set the CP 2/3 bit rate they shall be encoded as specified in Table A.7.

Table A.7 – Bit rate switch encoding

Bit rate	Switch setting
125 kbit/s	0
250 kbit/s	1
500 kbit/s	2

A.3.4.2.2 Labeling

The bit rate switches shall be labeled with one of the following:

“DR”;

“Data Rate”.

Bibliography

IEC/TR 61158-1 (Ed.2.0), *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61784-1 (Ed.2.0), *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*

ISO/IEC 8802 (all parts), *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks*

ISO/IEC 9314-2, *Information processing systems – Fibre Distributed Data Interface (FDDI) – Part 2: Token Ring Media Access Control (MAC)*

ANSI X3.66, *Advanced data communication control procedures (ADCCP)*

ANSI X3.159, *Information Systems – Programming Language C*

ANSI X3J16 / ISO WG21 *committee draft working paper for a C++ standard*

Internet Engineering Task Force (IETF), *Request for Comments (RFC)*:

RFC 791, *Internet Protocol*
(available at <<http://www.ietf.org/rfc/rfc0791.txt>>)

RFC 793, *Transmission Control Protocol*
(available at <<http://www.ietf.org/rfc/rfc0793.txt>>)

RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets : MIB-I* (available at <<http://www.ietf.org/rfc/rfc1213.txt>>)

RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types*
(available at <<http://www.ietf.org/rfc/rfc1643.txt>>)

IETF <draft-ietf-dhc-fqdn-option-12.txt>: February 24th, 2006, *The DHCP Client FQDN Option* (available at <<http://www.ietf.org/internet-drafts/draft-ietf-dhc-fqdn-option-12.txt>>)

ODVA: THE CIP NETWORKS LIBRARY – *Volume 1: Common Industrial Protocol (CIPTM) – Edition 3.1, November 2006*, available at <<http://www.odva.org>>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 2: EtherNet/IP™ Adaptation of CIP – Edition 1.3, November 2006*, available at <<http://www.odva.org>>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 3: DeviceNet™ Adaptation of CIP – Edition 1.3, November 2006*, available at <<http://www.odva.org>>

ControlNet International: THE CIP NETWORKS LIBRARY – *Volume 4: ControlNet™ Adaptation of CIP – Edition 1.1, December 2005*, available at <<http://www.controlnet.org>>

INDEX

Addressing

service access points

single (DLSAP-address), **16**single or group of (DL(SAP)-address), **15**

DL(SAP)-address. See Addressing, service access points, single or group

DLSAP, **15**

DLSAP-address. See Addressing, service access points, single

Link

extended, **16**local, **15**

Type 2

DLPDU, **18**fixed tag, **18**generic tag, **18**guardband, **18**moderator, **19, 20**

Withdrawing
IECNORM.COM: Click to view the full PDF of IEC 61158-4-2:2007

SOMMAIRE

AVANT-PROPOS.....	225
INTRODUCTION.....	228
1 Domaine d'application	229
1.1 Généralités.....	229
1.2 Spécifications	229
1.3 Modes opératoires	229
1.4 Applicabilité.....	230
1.5 Conformité	230
2 Références normatives	230
3 Termes, définitions, symboles et abréviations	231
3.1 Termes et définitions du modèle de référence	231
3.2 Termes et définitions de convention de service	233
3.3 Termes et définitions communs.....	234
3.4 Définitions de type 2 supplémentaires.....	236
3.5 Symboles et abréviations de type 2.....	243
4 Présentation du protocole DL.....	244
4.1 Généralités.....	244
4.2 Services fournis par la DL.....	247
4.3 Structure et définition des adresses DL.....	247
4.4 Service supposé provenir de PhL.....	249
4.5 Classes fonctionnelles.....	252
5 Structure générale et codage des PhLDU et DLPDU et éléments de procédure connexes	252
5.1 Présentation.....	252
5.2 Mode opératoire d'accès au support.....	253
5.3 Structure et codage de DLPDU.....	257
5.4 Composants Lpacket	262
5.5 Modes opératoires DLPDU.....	265
5.6 Récapitulatif des services de support et objets DLL.....	266
6 Structure DLPDU spécifique, codage et procédures	268
6.1 Langage de modélisation	268
6.2 Services utilisateur DLS.....	270
6.3 Lpacket à balise générique	277
6.4 Lpacket modérateur.....	277
6.5 Lpacket de répartition temporelle	279
6.6 Lpacket UCMM.....	281
6.7 Lpacket Keeper UCMM	282
6.8 Lpacket TUI.....	282
6.9 Paramètres de liaison Lpacket et tMinus Lpacket	284
6.10 Lpacket I'm-alive	285
6.11 Lpackets ping	288
6.12 Lpacket WAMI	289
6.13 Lpacket Debug	290
6.14 Lpacket IP.....	290
6.15 Lpacket Ethernet	290
7 Objets de gestion de la station.....	291

7.1	Généralités.....	291
7.2	Objet ControlNet.....	292
7.3	Objet Keeper	303
7.4	Objet de planification	333
7.5	Objet d'interface TCP/IP	344
7.6	Objet de liaison Ethernet	356
7.7	Objet DeviceNet	362
7.8	Objet de configuration de connexion (CCO).....	370
8	Autre éléments de procédure DLE.....	392
8.1	Moniteur de connexion au réseau (NAM).....	392
8.2	Calcul des paramètres de liaison	400
9	Spécification détaillée des composants DL.....	407
9.1	Généralités.....	407
9.2	Machine de contrôle d'accès (ACM)	407
9.3	TxLLC	425
9.4	RxLLC.....	429
9.5	Machine de transmission (TxM).....	432
9.6	Machine destinataire (RxM)	435
9.7	Convertisseur parallèle-série	441
9.8	Convertisseur série-parallèle	442
9.9	Gestion de DLL	443
Annexe A (normative)	Voyants et commutateurs.....	446
A.1	Objectif	446
A.2	Voyants.....	446
A.3	Commutateurs.....	459
Bibliographie		461
INDEX		462
Figure 1	– Relations des DLSAP, des adresses DLSAP et des adresses DL de groupe	235
Figure 2	– Architecture interne de la couche de liaison de données	246
Figure 3	– Structure de base d'une adresse MAC ID.....	248
Figure 4	– Structure de base d'une adresse de balise générique.....	248
Figure 5	– Structure de base d'une adresse de balise fixe	249
Figure 6	– M ₁ symbols et codage Manchester à 5 MHz (informative)	250
Figure 7	– Structure de la NUT.....	254
Figure 8	– Accès au support pendant la durée planifiée	255
Figure 9	– Accès au support pendant la durée non planifiée	256
Figure 10	– Format de DLPDU	258
Figure 11	– Annulation d'une DLPDU pendant la transmission	262
Figure 12	– Format de Lpacket.....	262
Figure 13	– Format Lpacket à balise générique	264
Figure 14	– Format Lpacket à balise fixe	264
Figure 15	– Paramètre goodness de TimeDist_Lpacket.....	280
Figure 16	– Exemple d'algorithme de traitement l'm alive.....	288
Figure 17	– Algorithme CRC de l'objet Keeper.....	311
Figure 18	– Diagramme d'états d'activation de l'objet Keeper	326

Figure 19 – Diagramme d'états de fonctionnement de l'objet Keeper.....	328
Figure 20 – Traitement de la modification de réseau synchronisée.....	333
Figure 21 – Diagramme de transition d'états de l'objet d'interface TCP/IP	355
Figure 22 – Diagramme d'édition de l'objet de configuration de connexion	392
Figure 23 – Diagramme d'état du NAM	393
Figure A.1 – Étiquetage du voyant d'état du réseau non redondant.....	451
Figure A.2 – Étiquetage du voyant d'état du réseau redondant.....	452
Tableau 1 – Composants de la couche de liaison de données.....	245
Tableau 2 – Allocation d'adresses MAC ID	248
Tableau 3 – Définitions de service de balise fixe	249
Tableau 4 – Règles de codage des données	250
Tableau 5 – Symboles de données M.....	251
Tableau 6 – Tableau de vérité de ph_status_indication.....	252
Tableau 7 – Longueur, polynômes et constantes FCS.....	259
Tableau 8 – Services de support et objets DLL.....	266
Tableau 9 – Types de données élémentaires.....	270
Tableau 10 – Événements DLL	275
Tableau 11 – Priorité de répartition temporelle	280
Tableau 12 – Format du Lpacket TUI.....	283
Tableau 13 – Attributs de classe de l'objet ControlNet.....	292
Tableau 14 – Attributs d'instance de l'objet ControlNet.....	293
Tableau 15 – Bits de balise d'état TUI.....	297
Tableau 16 – Bits d'état du canal.....	299
Tableau 17 – Services communs de l'objet ControlNet.....	300
Tableau 18 – Services spécifiques à la classe de l'objet ControlNet.....	301
Tableau 19 – Historique de révision de l'objet Keeper.....	303
Tableau 20 – Attributs de classe de l'objet Keeper.....	304
Tableau 21 – Attributs d'instance de l'objet Keeper.....	305
Tableau 22 – Définitions de l'état de fonctionnement de l'objet Keeper	308
Tableau 23 – Définitions de bit de balise d'état du port.....	309
Tableau 24 – Bits de balise d'état TUI	310
Tableau 25 – Attributs Keeper.....	312
Tableau 26 – Exigences en matière de mémoire (en octets) des attributs Keeper	313
Tableau 27 – Services communs de l'objet Keeper	314
Tableau 28 – Services spécifiques à la classe de l'objet Keeper	315
Tableau 29 – Codes d'erreur du service	316
Tableau 30 – Format de virement du TUI Lpacket.....	321
Tableau 31 – Codes d'erreur du service	322
Tableau 32 – États de fonctionnement de l'objet Keeper.....	322
Tableau 33 – Matrice d'événement d'état de l'objet Keeper.....	329
Tableau 34 – Attributs de classe de l'objet de planification	334
Tableau 35 – Attributs d'instance de l'objet de planification	334

Tableau 36 – Services communs de l'objet de planification	335
Tableau 37 – Descriptions de l'erreur d'état de Create.....	336
Tableau 38 – Description d'erreur d'état pour Delete et Kick_Timer	336
Tableau 39 – Services spécifiques à la classe de l'objet de planification	337
Tableau 40 – Descriptions de l'erreur d'état de Read.....	339
Tableau 41 – Descriptions de l'erreur d'état de Conditional_Write.....	340
Tableau 42 – Descriptions de l'erreur d'état de Forced_Write	340
Tableau 43 – Descriptions de l'erreur d'état de Change_Start.....	341
Tableau 44 – Descriptions de l'erreur d'état de Break_Connections	342
Tableau 45 – Descriptions de l'erreur d'état de Change_Complete.....	342
Tableau 46 – Descriptions de l'erreur d'état de Restart_Connections.....	343
Tableau 47 – Attributs de classe de l'objet d'interface TCP/IP.....	345
Tableau 48 – Attributs d'instance de l'objet d'interface TCP/IP.....	345
Tableau 49 – Bits Etat	348
Tableau 50 – Bits de capacité de configuration.....	348
Tableau 51 – Bits de contrôle de configuration	349
Tableau 52 – Exemple de chemin.....	350
Tableau 53 – Composants de configuration d'interface	351
Tableau 54 – Valeurs d'alloc control.....	352
Tableau 55 – Services communs de l'objet d'interface TCP/IP	353
Tableau 56 – Format de réponse Get_Attribute_All.....	353
Tableau 57 – Historique de révision de l'objet de liaison Ethernet.....	356
Tableau 58 – Attributs de classe de l'objet de liaison Ethernet.....	356
Tableau 59 – Attributs d'instance de l'objet de liaison Ethernet.....	356
Tableau 60 – Bits des balises d'interface.....	359
Tableau 61 – Bits de contrôle.....	360
Tableau 62 – Services communs de l'objet de liaison Ethernet	361
Tableau 63 – Services spécifiques à la classe de l'objet de liaison Ethernet.....	362
Tableau 64 – Historique de révision de l'objet DeviceNet.....	362
Tableau 65 – Attributs de classe de l'objet DeviceNet.....	362
Tableau 66 – Attributs d'instance de l'objet DeviceNet.....	363
Tableau 67 – Valeurs de l'attribut Bit Rate.....	365
Tableau 68 – Valeurs de l'attribut BOI.....	366
Tableau 69 – Description de bit des compteurs de diagnostic	368
Tableau 70 – Services communs de l'objet DeviceNet	369
Tableau 71 – Paramètre du service Reset	369
Tableau 72 – Valeurs de paramètre du service Reset	369
Tableau 73 – Services spécifiques à la classe de l'objet DeviceNet	370
Tableau 74 – Historique de révision de l'objet de configuration de connexion.....	371
Tableau 75 – Attributs de classe de l'objet de configuration de connexion.....	371
Tableau 76 – Valeurs de numéro de format	373
Tableau 77 – Attributs d'instance de l'objet de configuration de connexion.....	373
Tableau 78 – Valeurs d'état de connexion de l'auteur	376

Tableau 79 – Valeurs d'état de connexion cible	376
Tableau 80 – Balises de connexion	377
Tableau 81 – Formats de mappage d'E-S	379
Tableau 82 – Services communs de l'objet de configuration de connexion	380
Tableau 83 – Codes d'erreur Get_Attribute_All	380
Tableau 84 – Réponse Get_Attribute_All	381
Tableau 85 – Codes d'erreur Set_Attribute_All	382
Tableau 86 – Demande Set_Attribute_All	383
Tableau 87 – Paramètres de demande Create	384
Tableau 88 – Codes d'erreur Create	385
Tableau 89 – Codes d'erreur Delete	385
Tableau 90 – Codes d'erreur Restore	385
Tableau 91 – Services spécifiques à la classe de l'objet de configuration de connexion	386
Tableau 92 – Codes d'erreur Kick_Timer	386
Tableau 93 – Codes d'erreur Open_Connection	387
Tableau 94 – Codes d'erreur Close_Connection	387
Tableau 95 – Codes d'erreur Stop_Connection	387
Tableau 96 – Codes d'erreur Change_Start	388
Tableau 97 – Paramètre du service Get_Status	388
Tableau 98 – Réponse du service Get_Status	389
Tableau 99 – Codes d'erreur du service Get_Status	389
Tableau 100 – Paramètre du service Change_Complete	390
Tableau 101 – Codes d'erreur du service Change_Complete	390
Tableau 102 – Paramètre du service Audit_Changes	390
Tableau 103 – Codes d'erreur du service Audit_Changes	390
Tableau 104 – États NAM	393
Tableau 105 – Paramètres de liaison par défaut	394
Tableau 106 – Caractéristiques de temporisation PhL	401
Tableau A.1 – Voyant d'état du module	447
Tableau A.2 – Voyants d'état du réseau	449
Tableau A.3 – Voyant d'état du réseau	453
Tableau A.4 – Voyant d'état du réseau	455
Tableau A.5 – Voyant d'état du module/réseau combiné	457
Tableau A.6 – Voyant d'état E-S	458
Tableau A.7 – Codage du commutateur de vitesse en bits	460

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

**RÉSEAUX DE COMMUNICATION INDUSTRIELS –
SPÉCIFICATIONS DES BUS DE TERRAIN –****Partie 4-2: Spécification du protocole de la couche liaison de données –
Éléments de Type 2****AVANT-PROPOS**

- 1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
- 2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
- 3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
- 4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
- 5) La CEI n'a prévu aucune procédure de marquage valant indication d'approbation et n'engage pas sa responsabilité pour les équipements déclarés conformes à une de ses Publications.
- 6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
- 7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
- 8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.

NOTE L'utilisation de certains des types de protocoles est restreinte par les détenteurs des droits de propriété intellectuelle correspondants. Quoi qu'il en soit, l'engagement pris par les détenteurs, quant à une diffusion limitée desdits droits de propriété intellectuelle, permet d'utiliser un type particulier de protocole de la Couche Liaison de données avec des protocoles de Couche Physique et de Couche Application dans les combinaisons de types explicitement spécifiées dans la série CEI 61784. L'utilisation de divers types de protocoles dans d'autres combinaisons peut nécessiter l'autorisation de leurs détenteurs de droits de propriétés intellectuelle respectifs.

La Commission Electrotechnique Internationale (CEI) attire l'attention sur le fait qu'il est déclaré que la conformité avec la présente norme peut impliquer l'utilisation de brevets présentés ci-après, dans lesquels la notation [xx] indique le détenteur du brevet:

Type 2 et autres types éventuels:

US 5,400,331	[RA]	Communication network interface with screeners for incoming messages
US 5,471,461	[RA]	Digital communication network with a moderator station election process
US 5,491,531	[RA]	Media access controller with a shared class message delivery capability
US 5,493,571	[RA]	Apparatus and method for digital communications with improved delimiter detection
US 5,537,549	[RA]	Communication network with time coordinated station activity by time slot and periodic interval number

US 5,553,095 [RA] Method and apparatus for exchanging different classes of data during different time intervals

La CEI ne prend pas position eu égard à la preuve, la validité et la portée de ces droits de propriétés.

Les détenteurs de ces droits de propriété ont donné l'assurance à la CEI qu'ils consentent à négocier des licences avec des demandeurs du monde entier, en des termes et à des conditions raisonnables et non discriminatoires. A ce propos, la déclaration des détenteurs des droits de propriétés est enregistrée à la CEI. Des informations peuvent être obtenues auprès de

[RA]: Rockwell Automation, Inc.
1201 S. Second Street
Milwaukee, WI 53204
USA
Attention: Intellectual Property Dept.

L'attention est attirée sur le fait que certains des éléments de la présente norme peuvent faire l'objet de droits de propriété industrielle distincts de ceux mentionnés ci-dessus. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété ou de ne pas avoir signalé leur existence.

La Norme internationale CEI 61158-4-2 a été établie par le sous-comité 65C, Réseaux de communications industriels, du comité d'études 65 de la CEI, Mesure, commande et automation dans les processus industriels.

Cette première édition et ses parties d'accompagnement de la sous-série CEI 61158-4 annulent et remplacent la CEI 61158-4:2003. La présente édition de cette partie constitue une révision mineure. Cette partie et ses parties d'accompagnement pour le Type 2 annulent et remplacent également la CEI PAS 62413, publiée en 2005.

Cette édition de la CEI 61158-4 inclut les modifications majeures suivantes par rapport à l'édition précédente:

- a) suppression du précédent bus de terrain de Type 6 et de la référence à une couche de liaison de données de bus de terrain de Type 5, en raison du manque d'adéquation au marché;
- b) ajout de nouveaux types de bus de terrain;
- c) division de cette partie en parties multiples numérotées -4-1, -4-2, ..., -4-19.

La présente version bilingue (2014-07) correspond à la version anglaise monolingue publiée en 2007-12.

Le texte anglais de cette norme est issu des documents 65C/474/FDIS et 65C/485/RVD.

Le rapport de vote 65C/485/RVD donne toute information sur le vote ayant abouti à l'approbation de cette norme.

La version française de cette norme n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous <http://webstore.iec.ch> dans les données relatives à la publication recherchée. A cette date, la publication sera:

- reconduite;
- supprimée;
- remplacée par une édition révisée, ou
- amendée.

NOTE La révision de la présente norme sera synchronisée avec les autres parties de la série CEI 61158.

La liste de toutes les parties de la série CEI 61158, publiée sous le titre général *Réseaux de communication industriels – Spécifications des bus de terrain*, est disponible sur le site web de la CEI.

Withdrawing
IECNORM.COM: Click to view the full PDF of IEC 61158-4-2:2007

INTRODUCTION

La présente partie de la CEI 61158 appartient à la série de normes visant à faciliter l'interconnexion des composants du système d'automatisation. Elle est liée aux autres normes de la série telle que définie par le modèle de référence de bus de terrain « à trois couches » décrits dans la CEI/TR 61158-1.

Le protocole de liaison de données assure un service de liaison de données en s'appuyant sur les services offerts par la couche physique. La présente norme a pour principal objet de préciser un ensemble de règles de communication, exprimées sous la forme de modes opératoires que doivent réaliser des entités de liaison de données homologues (DLE) au moment de la communication. Ces règles de communication ont pour vocation de fournir une base de développement stable visant à atteindre différents objectifs:

- a) guider les implémenteurs et les concepteurs;
- b) réaliser les essais et acquérir l'équipement;
- c) dans le cadre d'un accord d'intégration des systèmes dans l'environnement de systèmes ouverts;
- d) dans le cadre d'une meilleure compréhension des communications à contrainte de temps au sein de l'OSI.

La présente norme porte en particulier sur la communication et l'interfonctionnement des capteurs, des effecteurs et d'autres dispositifs d'automatisation. Grâce à la présente norme associée à d'autres normes des modèles de référence OSI ou de bus de terrain, des systèmes par ailleurs incompatibles peuvent fonctionner ensemble, quelle que soit leur combinaison.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 4-2: Spécification du protocole de la couche liaison de données – Éléments de Type 2

1 Domaine d'application

1.1 Généralités

La couche de liaison de données assure les communications de messagerie à contrainte de temps de base entre les dispositifs d'un environnement d'automatisation.

Ce protocole offre des opportunités de communication séquentielle et synchrone cyclique à toutes les entités de liaison de données participantes. Un accès planifié de premier plan est proposé à toutes les activités à contrainte de temps, un accès non planifié l'étant aux activités moins critiques.

Des transferts déterministes et synchronisés peuvent être assurés à des intervalles cycliques jusqu'à 1 ms pour des dispositifs distants de 25 km. Cette performance peut être adaptée de manière dynamique et en ligne en reconfigurant les paramètres de la liaison locale sans interrompre le fonctionnement normal. De la même manière, des connexions DL et de nouveaux dispositifs peuvent être ajoutés ou retirés pendant le fonctionnement normal.

Ce protocole offre la possibilité de maintenir la synchronisation d'horloge d'une liaison étendue à 10 μ s près.

Ce protocole permet d'optimiser chaque opportunité d'accès en concaténant plusieurs DLSDU et DLPCI associés en une seule DLPDU, améliorant le transfert de données des entités de liaison de données qui émettent activement plusieurs flux de données.

La taille maximale du système est un nombre illimité de liaisons de 99 nœuds, comportant chacune 255 adresses DLSAP. Chaque liaison comporte un maximum de 2^{24} homologues connexes et DLCEP d'éditeur.

1.2 Spécifications

La présente norme spécifie

- les modes opératoires de transfert opportun des données et des informations de commande entre une entité utilisateur de liaison de données et une entité utilisateur homologue, et parmi les entités de liaison de données formant le fournisseur de service de liaison de données distribué;
- la structure des DLPDU de bus de terrain utilisée par le protocole de la présente norme pour le transfert des données et des informations de commande, et leur représentation sous forme d'unités de données d'interface physique.

1.3 Modes opératoires

Les modes opératoires sont définis en termes:

- d'interactions entre les entités DL (DLE) homologues par l'échange de DLPDU de bus de terrain;
- d'interactions entre un fournisseur de service DL (DLS) et un utilisateur DLS au sein du même système par l'échange de primitives DLS;

- c) d'interactions entre un fournisseur DLS et un fournisseur de service Ph au sein du même système par l'échange de primitives de service Ph.

1.4 Applicabilité

Ces modes opératoires s'appliquent aux instances de communication entre des systèmes qui prennent en charge des services de communications à contrainte de temps dans la couche de liaison de données des modèles de référence OSI ou de bus de terrain, et qui peuvent être connectés dans un environnement d'interconnexion de systèmes ouverts.

Les profils sont un moyen simple à plusieurs attributs de récapituler les capacités d'une mise en œuvre, et donc son applicabilité en fonction des différents besoins de communications à contrainte de temps.

1.5 Conformité

La présente norme spécifie également les exigences relatives aux systèmes mettant en œuvre ces modes opératoires. La présente norme ne comporte aucun essai visant à démontrer la conformité à ces exigences.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC 61131-3, *Programmable controllers – Part 3: Programming languages* (disponible en anglais uniquement)

IEC 61158-2 (Ed.4.0), *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition* (disponible en anglais uniquement)

IEC 61158-3-2, *Industrial communication networks – Fieldbus specifications – Part 3-2: Data-link layer service definition – Type 2 elements* (disponible uniquement en anglais)

IEC 61158-5-2, *Industrial communication networks – Fieldbus specifications – Part 5-2: Application layer service definition – Type 2 elements* (disponible uniquement en anglais)

IEC 61158-6-2, *Industrial communication networks – Fieldbus specifications – Part 6-2: Application layer protocol specification – Type 2 elements* (disponible uniquement en anglais)

IEC 61784-3-2, *Industrial communication networks – Profiles – Part 3-2: Functional safety fieldbuses – Additional specifications for CPF 2* (disponible uniquement en anglais)

IEC 62026-3¹, *Low-voltage switchgear and controlgear – Controller-device interfaces (CDIs) – Part 3: DeviceNet* (disponible uniquement en anglais)

ISO/CEI 3309, *Technologies de l'information – Télécommunications et échange d'informations entre systèmes – Procédures de commande de liaison de données à haut niveau (HDLC) – Structure de trame*

ISO/CEI 7498-1, *Technologies de l'information – Modèle de référence de base pour l'interconnexion de systèmes ouverts (OSI) Modèle de base*

¹ A publier.

ISO/CEI 7498-3, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Dénomination et adressage*

ISO/CEI 8802-3: *Technologies de l'information – Télécommunications et échange d'informations entre systèmes - Réseaux locaux et métropolitains - Exigences spécifiques - Partie 3: Accès multiple par surveillance du signal et détection de collision (CSMA/CD) et spécifications pour la couche physique*

ISO/CEI 10731, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base – Conventions pour la définition des services OSI*

ISO 11898:1993², *Véhicules routiers – Échange d'information numérique – Gestionnaire de réseau de communication à vitesse élevée (CAN)*

3 Termes, définitions, symboles et abréviations

Pour les besoins du présent document, les termes, définitions, symboles et abréviations suivants s'appliquent.

3.1 Termes et définitions du modèle de référence

La présente norme repose en partie sur les concepts développés dans l'ISO/CEI 7498-1 et l'ISO/CEI 7498-3, et utilise les termes suivants.

3.1.1	adresse DL appelée	[ISO/CEI 7498-3]
3.1.2	adresse DL appelante	[ISO/CEI 7498-3]
3.1.3	connexion centralisée à plusieurs extrémités	[ISO/CEI 7498-1]
3.1.4	entités (N)correspondantes entités DL correspondantes (N=2) entités Ph correspondantes (N=1)	[ISO/CEI 7498-1]
3.1.5	démultiplexage	[ISO/CEI 7498-1]
3.1.6	adresse DL	[ISO/CEI 7498-3]
3.1.7	mappage d'adresse DL	[ISO/CEI 7498-1]
3.1.8	connexion DL	[ISO/CEI 7498-1]
3.1.9	extrémité de connexion DL	[ISO/CEI 7498-1]
3.1.10	identifiant d'extrémité de connexion DL	[ISO/CEI 7498-1]
3.1.11	transmission en mode connexion DL	[ISO/CEI 7498-1]
3.1.12	transmission en mode sans connexion DL	[ISO/CEI 7498-1]
3.1.13	collecteur de données DL	[ISO/CEI 7498-1]
3.1.14	source de données DL	[ISO/CEI 7498-1]
3.1.15	transmission duplex DL	[ISO/CEI 7498-1]
3.1.16	installation DL	[ISO/CEI 7498-1]

² Une édition plus récente de la présente norme a été publiée, mais seule l'édition citée s'applique.

3.1.17	vue locale DL	[ISO/CEI 7498-3]
3.1.18	nom DL	[ISO/CEI 7498-3]
3.1.19	protocole DL	[ISO/CEI 7498-1]
3.1.20	identifiant de connexion de protocole DL	[ISO/CEI 7498-1]
3.1.21	informations de commande de protocole DL	[ISO/CEI 7498-1]
3.1.22	unité de données de protocole DL	[ISO/CEI 7498-1]
3.1.23	identifiant de version de protocole DL	[ISO/CEI 7498-1]
3.1.24	relais DL	[ISO/CEI 7498-1]
3.1.25	identifiant de connexion de service DL	[ISO/CEI 7498-1]
3.1.26	unité de données de service DL	[ISO/CEI 7498-1]
3.1.27	transmission simplex DL	[ISO/CEI 7498-1]
3.1.28	sous-système DL	[ISO/CEI 7498-1]
3.1.29	données utilisateur DL	[ISO/CEI 7498-1]
3.1.30	contrôle de flux	[ISO/CEI 7498-1]
3.1.31	gestion de couche	[ISO/CEI 7498-1]
3.1.32	multiplexage	[ISO/CEI 7498-3]
3.1.33	autorité (d'adressage) de dénomination	[ISO/CEI 7498-3]
3.1.34	domaine (d'adressage) de dénomination	[ISO/CEI 7498-3]
3.1.35	sous-domaine (d'adressage) de dénomination	[ISO/CEI 7498-3]
3.1.36	entité (N) entité DL entité Ph	[ISO/CEI 7498-1]
3.1.37	unité de données d'interface (N) unité de données de service DL (N=2) unité de données d'interface Ph (N=1)	[ISO/CEI 7498-1]
3.1.38	couche (N) couche DL (N=2) couche Ph (N=1)	[ISO/CEI 7498-1]
3.1.39	service (N) service DL (N=2) service Ph (N=1)	[ISO/CEI 7498-1]
3.1.40	point d'accès au service (N) point d'accès au service DL (N=2) point d'accès au service Ph (N=1)	[ISO/CEI 7498-1]
3.1.41	adresse de point d'accès au service (N) adresse de point d'accès au service DL (N=2) adresse de point d'accès au service Ph (N=1)	[ISO/CEI 7498-1]
3.1.42	entités homologues	[ISO/CEI 7498-1]

3.1.43	informations de contrôle d'interface Ph	[ISO/CEI 7498-1]
3.1.44	données d'interface Ph	[ISO/CEI 7498-1]
3.1.45	nom primitif	[ISO/CEI 7498-3]
3.1.46	réassemblage	[ISO/CEI 7498-1]
3.1.47	recombinaison	[ISO/CEI 7498-1]
3.1.48	réinitialisation	[ISO/CEI 7498-1]
3.1.49	adresse DL de réponse	[ISO/CEI 7498-3]
3.1.50	acheminement	[ISO/CEI 7498-1]
3.1.51	segmentation	[ISO/CEI 7498-1]
3.1.52	séquencement	[ISO/CEI 7498-1]
3.1.53	éclatement	[ISO/CEI 7498-1]
3.1.54	nom synonyme	[ISO/CEI 7498-3]
3.1.55	gestion-système	[ISO/CEI 7498-1]

3.2 Termes et définitions de convention de service

La présente norme utilise également les termes définis dans l'ISO/CEI 10731 tels qu'ils s'appliquent à la couche de liaison de données:

3.2.1	accepteur
3.2.2	service asymétrique
3.2.3	confirm (primitive); requestor.deliver (primitive)
3.2.4	deliver (primitive)
3.2.5	DL-confirmed-facility
3.2.6	DL-facility
3.2.7	DL-local-view
3.2.8	DL-mandatory-facility
3.2.9	DL-non-confirmed-facility
3.2.10	DL-provider-initiated-facility
3.2.11	DL-provider-optional-facility
3.2.12	DL-service-primitive; primitive
3.2.13	DL-service-provider
3.2.14	DL-service-user
3.2.15	DL-user-optional-facility
3.2.16	indication (primitive) acceptor.deliver (primitive)

3.2.17 multi-peer (multi-homologue)

**3.2.18 demande (primitive);
requestor.submit (primitive)**

3.2.19 demandeur

**3.2.20 réponse (primitive);
acceptor.submit (primitive)**

3.2.21 submit (primitive)

3.2.22 service symétrique

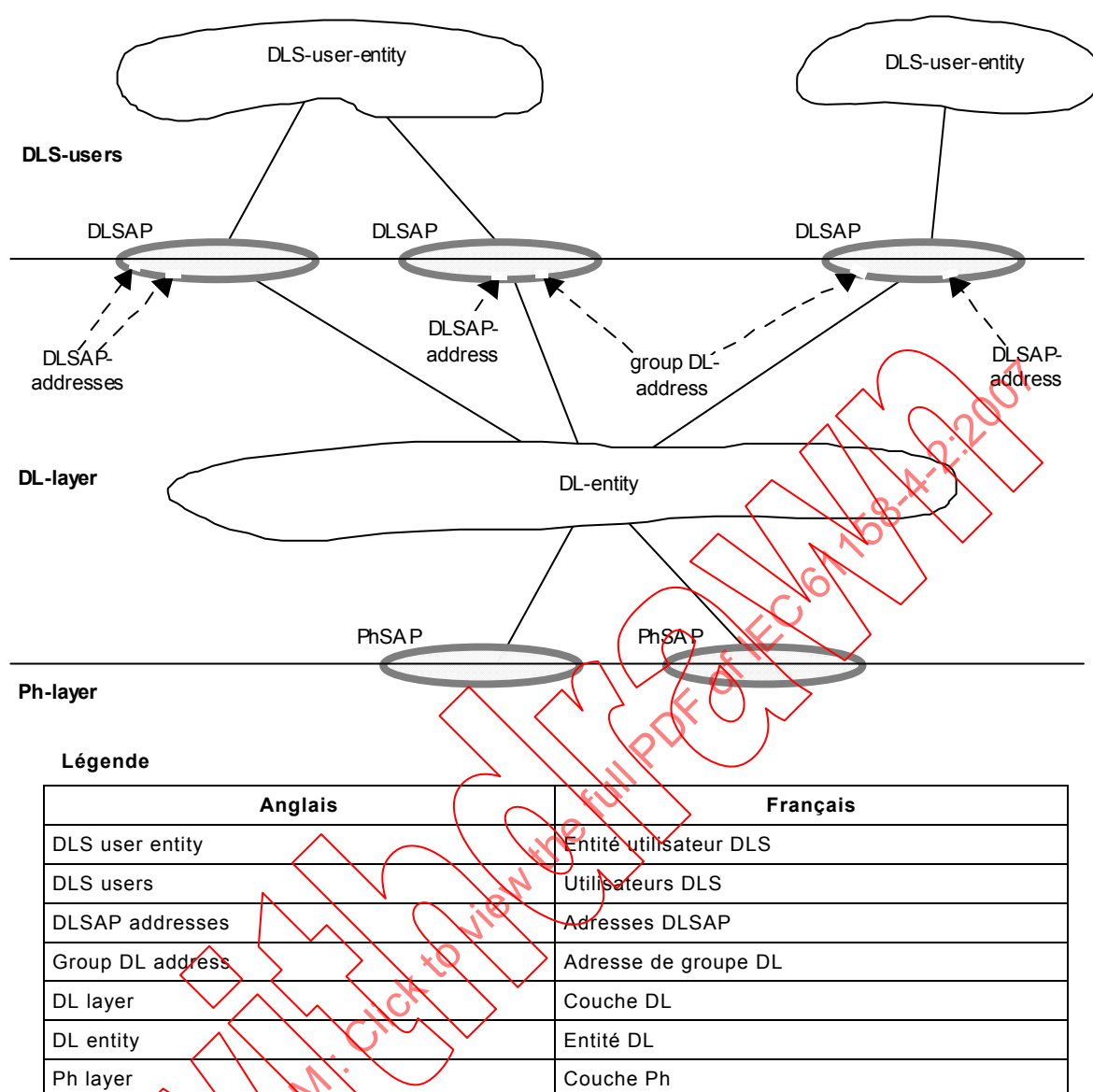
3.3 Termes et définitions communs

NOTE La plupart des définitions sont communes à plusieurs types de protocole et ne sont pas nécessairement utilisées par chacun d'eux.

3.3.1

segment DL, liaison, liaison locale

sous-réseau DL dans lequel l'une des DLE connectées peut communiquer directement sans l'intervention d'un relais DL, à chaque fois que toutes les DLE participant à une instance de communication sont simultanément attentives au sous-réseau DL pendant la/les période(s) de tentative de communication



NOTE 1 Les DLSAP et PhSAP sont décrits comme des éléments ovales à cheval sur la limite entre deux couches adjacentes.

NOTE 2 Les adresses DL sont décrites comme désignant de petits intervalles (points d'accès) dans la partie DLL d'une DLSAP.

NOTE 3 Une seule entité DL peut comporter plusieurs adresses DLSAP et adresses DL de groupe associées à une seule DLSAP.

Figure 1 – Relations des DLSAP, des adresses DLSAP et des adresses DL de groupe

3.3.2 DLSAP

point distinctif au niveau duquel des services DL sont fournis par une seule entité DL à une seule entité de couche supérieure.

NOTE Cette définition, déduite de l'ISO/CEI 7498-1, est reprise ici pour faciliter la compréhension de la distinction critique entre les DLSAP et leurs adresses DL. (Voir Figure 1.)

3.3.3 adresse DL(SAP)

adresse DLSAP individuelle désignant le DLSAP d'un utilisateur DLS unique ou adresse DL de groupe désignant plusieurs DLSAP, chacun d'eux appartenant à un seul utilisateur DLS.

NOTE Cette terminologie a été choisie car l'ISO/CEI 7498-3 ne permet pas d'utiliser le terme adresse DLSAP pour désigner plusieurs DLSAP uniques d'un seul utilisateur DLS.

3.3.4

adresse DLSAP (individuelle)

adresse DL qui désigne une seul DLSAP dans la liaison étendue

NOTE Une seule entité DL peut comporter plusieurs adresses DLSAP associées à un seul DLSAP.

3.3.5

liaison étendue

sous-réseau DL composé d'un nombre maximal de liaisons interconnectées par des relais DL et partageant un espace de noms DL unique (adresse DL) dans lequel toutes les entités DL connectées peuvent communiquer, soit directement soit à l'aide d'une ou de plusieurs de ces entités relais DL

NOTE Une liaison étendue peut être composée d'une seule liaison.

3.3.6

trame

synonyme discrédité de DLPDU

3.3.7

adresse DL de groupe

adresse DL qui désigne potentiellement plusieurs DLSAP dans la liaison étendue. Une entité DL unique peut comporter plusieurs adresses DL de groupe avec un seul DLSAP. Une entité DL unique peut également comporter une seule adresse DL de groupe avec plusieurs DLSAP

3.3.8

nœud

entité DL unique telle qu'elle se présente sur une liaison locale

3.3.9

utilisateur DLS destinataire

utilisateur du service DL auquel sont destinées les données utilisateur DL

NOTE Un utilisateur de service DL peut être à la fois un utilisateur DLS expéditeur et destinataire.

3.3.10

utilisateur DLS expéditeur

utilisateur du service DL à la source des données utilisateur DL

3.4 Définitions de Type 2 supplémentaires

3.4.1

intervalle réel entre paquets (API, en anglais *Actual Packet Interval*)

mesure de la fréquence de génération de données Lpacket par une connexion particulière

3.4.2

allouer

prendre une ressource dans une zone commune et l'attribuer à l'utilisation exclusive d'une entité particulière

3.4.3

application

fonction ou structure de données pour lesquelles les données sont utilisées ou générées

3.4.4

objets d'application

classes contenant plusieurs objets permettant de gérer et d'assurer l'échange dynamique des messages sur le réseau et au sein du dispositif

3.4.5

attribut

description d'une caractéristique visible en externe ou d'une fonction d'un objet

NOTE Les attributs d'un objet contiennent des informations relatives aux parties variables d'un objet. En règle générale, ils donnent des informations d'état ou régissent le fonctionnement d'un objet. Les attributs peuvent également avoir un impact sur le comportement d'un objet. Les attributs sont divisés en attributs de classe et en attributs d'instance.

3.4.6

comportement

indication de la réponse d'un objet à des événements particuliers. Cette description comprend les relations entre les valeurs d'attribut et les services

3.4.7

bit

unité d'informations composée de 1 ou de 0. Il s'agit de la plus petite unité de données pouvant être transmise

3.4.8

blocage ou temps de blocage

durée nécessaire après la transmission avant que le nœud ne soit autorisé à recevoir des données

3.4.9

client

- 1) Objet qui utilise les services d'un autre objet (serveur) pour exécuter une tâche
- 2) Émetteur d'un message auquel un serveur réagit

3.4.10

objets de communication

composants permettant de gérer et d'assurer l'échange dynamique des messages sur le réseau (objet ControlNet ou UCMM (Unconnected Message Manager), par exemple)

3.4.11

connexion

liaison logique entre deux objets d'application de dispositifs identiques ou différents

3.4.12

ID de connexion (CID, Connection ID)

identifiant attribué à une transmission et associé à une connexion particulière entre des producteurs et des consommateurs, permettant d'identifier une partie spécifique des informations d'application

NOTE Dans la liaison de données, il s'agit d'une balise générique.

3.4.13

contrôle de redondance cyclique (CRC)

valeur résiduelle calculée à partir d'une matrice de données et utilisée comme signature représentative de la matrice

3.4.14

cyclique

terme utilisé pour décrire des événements qui se répètent de manière régulière et répétitive

3.4.15

surdité

situation dans laquelle le nœud n'entend pas la DLPDU modérateur, mais entend le trafic de liaison

3.4.16

dispositif

connexion matérielle physique à la liaison

NOTE Un dispositif peut contenir plusieurs nœuds.

3.4.17

DLPDU

unité de données de protocole de liaison de données (*data-link protocol data unit*)

NOTE Une DLPDU est composée d'un MAC ID source, d'aucun ou de plusieurs Lpackets et d'un FCS tel que transmis ou reçu par un PhE associé.

3.4.18

délimiteur de fin

ensemble unique de M_symbols identifiant la fin d'une DLPDU

3.4.19

erreur

écart ou discordance entre une valeur ou une condition calculée, observée ou mesurée et la valeur ou la condition spécifiée ou théoriquement correcte

3.4.20

liaison étendue

liaisons connectées par des routeurs DL (parfois appelées réseau local)

3.4.21

erreur FCS

erreur qui se produit lorsque la valeur de la séquence de contrôle de trame calculée après réception de tous les octets d'une DLPDU ne correspond pas à la valeur résiduelle prévue

3.4.22

balise fixe

identifiant (balise) à deux octets qui identifie un service spécifique que doit réaliser

- a) le nœud destinataire sur la liaison locale qui a un MAC ID spécifié, ou
- b) tous les nœuds destinataires de la liaison locale

NOTE L'identification du/des nœud(s) cible est comprise dans la balise à deux octets.

3.4.23

trame

synonyme discrédité de DLPDU

3.4.24

balise générique

identifiant (balise) à trois octets qui identifie une partie spécifique des informations

3.4.25

bande de garde

intervalle de temps attribué pour la transmission de la DLPDU modérateur

3.4.26

jeton implicite

mécanisme régissant les droits de transmission

NOTE Aucun message de jeton réel n'est transmis sur le support. Chaque nœud garde une trace du MAC ID du nœud qu'il suppose détenir les droits de transmission. Les droits de transmission sont transmis d'un nœud à l'autre en conservant un enregistrement du dernier nœud qui les a transmis. Un intervalle de temps est utilisé afin d'ignorer un nœud manquant dans la rotation.

3.4.27**registre de jeton implicite**

registre contenant le MAC ID du nœud détenant les droits de transmission

3.4.28**instance**

occurrence physique réelle d'un objet dans une classe, identifiant l'un des objets dans la même classe d'objets

EXEMPLE Californie est une instance de la classe d'objet État.

NOTE Les termes objet, instance et instance d'objet font référence à une instance particulière.

3.4.29**attribut d'instance**

attribut unique d'une instance d'objet, qui n'est pas partagé par la classe d'objets

3.4.30**instancié**

objet créé dans un dispositif

3.4.31**keeper**

objet chargé de distribuer les données de configuration de liaison à tous les nœuds de la liaison

3.4.32**liaison**

ensemble de nœuds dotés de MAC ID uniques, associés à des segments Ph connectés par des répéteurs Ph

3.4.33**petit-boutiste**

modèle d'organisation de mémoire et d'organisation de la transmission permettant respectivement de stocker l'octet de poids faible à l'adresse la plus basse et de le placer en premier sur le support

3.4.34**Lpacket**

sous-partie bien définie d'une DLPDU composée

- a) d'informations de taille et de commande
- b) d'une balise fixe ou générique, et
- c) de données utilisateur DLS ou, si la balise porte une signification DL, de données DL

3.4.35**M_symbol(s)**

symboles représentant les bits de données et les informations connexes codés et transmis par la PhL

3.4.36**nœud planifié maximal**

nœud doté du MAC ID le plus élevé pouvant utiliser un temps de référence sur une liaison

3.4.37**nœud non planifié maximal**

nœud doté du MAC ID le plus élevé pouvant utiliser un temps non planifié sur une liaison

3.4.38

routeur de message

objet d'un nœud qui distribue les demandes de message aux objets d'application concernés

3.4.39

modérateur

nœud doté du MAC ID le plus bas chargé de transmettre la DLPDU modérateur

3.4.40

DLPDU modérateur

DLPDU transmis par le nœud doté du MAC ID le plus bas afin de synchroniser les nœuds et de distribuer les paramètres de configuration de liaison

3.4.41

connexion multipoint

connexion entre un nœud et plusieurs autres nœuds

NOTE Les connexions multipoint permettent à plusieurs nœuds de consommateur (abonné) de recevoir des messages provenant d'un seul producteur (éditeur).

3.4.42

réseau

série de nœuds connectés par un certain type de support de communication. Les chemins de connexion entre une paire de nœuds peuvent contenir des répéteurs, des routeurs et des passerelles

3.4.43

moniteur d'accès au réseau (NAM, *Network Access Monitor*)

composant d'un nœud qui gère la communication avec un nœud temporaire connecté au NAP local des nœuds

3.4.44

port d'accès au réseau (NAP, *Network Access Port*)

variante de PhL permettant de connecter un nœud temporaire à la liaison par l'intermédiaire du NAP d'un nœud permanent

3.4.45

adresse réseau ou adresse de nœud

adresse du nœud sur la liaison (également appelée MAC ID)

3.4.46

voyants d'état du réseau

voyants d'un nœud signalant l'état des couches de liaison physiques et des couches de liaison de données

3.4.47

durée de mise à jour du réseau (NUT, *Network Update Time*)

intervalle de temps répétitif au cours duquel les données peuvent être envoyées sur la liaison

3.4.48

nœud

connexion logique à une liaison locale, nécessitant un MAC ID unique

NOTE Un seul dispositif physique peut apparaître sous la forme de plusieurs nœuds sur la même liaison locale. Pour les besoins de ce protocole, chaque nœud est considéré comme une DLE distincte.

3.4.49

non-concurrence

situation dans laquelle une transmission est reçue d'un MAC ID imprévu semblant enfreindre le protocole d'accès temporel

NOTE Cela peut se produire lorsqu'une connexion est établie entre deux liaisons de fonctionnement qui ne sont pas synchronisées, mais qui détiennent les mêmes informations de configuration.

3.4.50

symbole sans données

symbole PhL qui enfreint les exigences du codage Manchester biphasé L

3.4.51

objet

représentation abstraite d'un composant particulier d'un dispositif

NOTE Un objet peut être

- 1) une représentation abstraite des capacités d'un ordinateur. Les objets peuvent être composés de tout ou partie des composants suivants:
 - a) données (informations évoluant dans le temps);
 - b) configuration (paramètres de comportement);
 - c) méthodes (actions qui peut être réalisée à l'aide de données et d'une configuration).
- 2) un ensemble de données (se présentant sous la forme de variables) et de méthodes (modes opératoires) connexes d'utilisation de ces données qui définit clairement l'interface et le comportement.

3.4.52

service spécifique à l'objet

service défini par une classe d'objets particulière permettant d'exécuter une fonction requise qu'un service classique ne peut assurer

NOTE Un service spécifique à l'objet est unique à la classe d'objets qu'il définit.

3.4.53

auteur

client chargé d'établir un chemin de connexion vers la cible

3.4.54

nœud permanent

nœud dont la connexion au réseau n'utilise par la variante PhL du port d'accès au réseau (NAP)

NOTE Ce nœud peut éventuellement prendre en charge une variante PhL du NAP afin de permettre aux nœuds temporaires de se connecter au réseau.

3.4.55

produire

acte d'envoi de données à un consommateur

3.4.56

producteur

nœud chargé de l'envoi des données

3.4.57

support redondant

système s'appuyant sur plusieurs supports pour prévenir les défaillances de communication

3.4.58

intervalle demandé entre les paquets (RPI, *Requested Packet Interval*)

mesure de la fréquence de demande de transmission de Lpackets par l'application d'origine ou de paquets de données provenant de l'application cible

3.4.59

suspect

nœud ayant reçu une DLPDU modérateur opposé à la configuration de liaison qu'il utilise

3.4.60

planifié

transfert de données déterministe et répétable sur des durées de mise à jour du réseau prédéfinies

3.4.61

serveur

objet qui offre des services à un autre objet (client)

3.4.62

service

opération ou fonction qu'un objet et/ou une classe d'objets exécute à la demande d'un autre objet et/ou une autre classe d'objets

NOTE Un ensemble de services communs est défini et des dispositions pour la définition de services spécifiques à l'objet sont fournies. Les services spécifiques à l'objet sont définis par une classe d'objets particulière pour exécuter une fonction requise qui ne l'est pas par un service commun.

3.4.63

intervalle de temps

durée maximale requise pour détecter une transmission prévue

NOTE Chaque nœud attend un intervalle de temps pour chaque nœud manquant pendant la transmission du jeton impliqué.

3.4.64

délimiteur de début

ensemble unique de M_symbols identifiant le début d'une DL PDU

3.4.65

supernode

DLE dont l'ID MAC est nul

NOTE Cette adresse DL de la DLE est réservée à des fonctions DLL particulières.

3.4.66

identifiant unique de table (TUI, *Table Unique Identifier*)

code de référence unique utilisé par les objets ControlNet et Keeper pour faire référence à un ensemble cohérent d'attributs de configuration de liaison

3.4.67

balise

nom abrégé d'une partie spécifique des informations d'application, de deux ou trois octets de long

3.4.68

cible

nœud d'extrémité avec lequel une connexion est établie

3.4.69

nœud temporaire

synonyme de nœud transitoire

3.4.70

tMinus

nombre de NUT avant d'utiliser un nouvel ensemble de paramètres de configuration de liaison

3.4.71

tonalité

instant marquant la limite entre deux NUT

3.4.72**outil**

programme logiciel exécutable qui interagit avec l'utilisateur pour exécuter une fonction donnée

EXEMPLE Logiciel de planification de liaison (LSS).

3.4.73**ID de transaction**

zone d'un en-tête UCMM mettant une réponse en correspondance avec la demande associée, qu'un serveur répercute dans le message de réponse

3.4.74**nœud transitoire**

nœud ayant pour seul objet d'être connecté au réseau de manière temporaire à l'aide du support PhL NAP connecté au NAP d'un nœud permanent

3.4.75**Unconnected Message Manager (UCMM)**

composant d'un nœud qui transmet et reçoit des messages explicites non connectés et les envoie directement à l'objet Routeur de message

3.4.76**service non connecté**

service de messagerie qui ne dépend pas de la configuration d'une connexion entre des dispositifs pour autoriser les échanges d'informations

3.4.77**non planifié**

transfert de données qui utilise la durée attribuée restante dans le NUT à l'issue des transferts planifiés

3.5 Symboles et abréviations de Type 2

ACM	Machine de contrôle d'accès (<i>Access control machine</i>)
CA	Précision de l'horloge (<i>Clock Accuracy</i>)
CAN	Réseau local du contrôleur (<i>Controller Area Network</i>) [ISO 11898:1993]
COP	Mot de passe de l'auteur de la connexion (<i>Connection originator password</i>)
IP	Protocole Internet (<i>Internet Protocol</i>)
DEL	Diode ElectroLuminescente
LSS	Logiciel de planification de liaison (<i>Link Scheduling Software</i>)
MAC ID	Adresse MAC d'un nœud
NAM	Moniteur de connexion au réseau (<i>Network attachment monitor</i>)
NAP	Port d'accès au réseau (<i>Network Access Port</i>)
ND	Symbole sans données (<i>Non-data symbol</i>)
NUT	Durée de mise à jour du réseau (<i>Network Update Time</i>)
PT	Terminal de programmation (connexion au réseau temporaire) (<i>Programming terminal</i>)
Rcv	Recevoir
RPI	Intervalle demandé entre les paquets (<i>Requested Packet Interval</i>)
Rx	Réception
RxLLC	Contrôle de liaison logique de réception (<i>Receive logical link control</i>)
RxM	Machine destinataire
SEM	Matrice d'événement d'état (<i>State Event Matrix</i>)
SMAX	MAC ID du nœud planifié maximal

STD	Diagramme de transition d'états (<i>State transition diagram</i>), utilisé pour décrire le comportement de l'objet
TCP	Terminal Control Protocol
TUI	Identifiant unique de table (<i>Table Unique Identifier</i>)
Tx	Transmettre
TxLLC	Contrôle de liaison logique de transmission (<i>Transmit logical link control</i>)
TxM	Machine de transmission
UCCM	Unconnected Message Manager
UMAX	MAC ID du nœud non planifié maximal
USR	Registre de démarrage non planifié (<i>Unscheduled start register</i>)

4 Présentation du protocole DL

4.1 Généralités

4.1.1 Architecture DLL

La DLL de Type 2 est modélisée sous la forme d'une machine de contrôle d'accès (ACM) intégrée, dotée de fonctions de support de planification, et conçue pour assurer une prise en charge fiable et efficace des services de transfert de données en mode connexion de niveau supérieur et en mode sans connexion. L'interaction avec ces fonctions de niveau supérieur sont des fonctions de gestion de DLL.

La structure PhL et les délimiteurs sont gérés par les fonctions DLL pour la conversion parallèle-série et série-parallèle des demandes M_symbol et des indications.

Dans la couche de liaison de données, la machine de contrôle d'accès est chargée de détecter et de récupérer les interruptions de réseau. L'ACM a pour principaux objectifs:

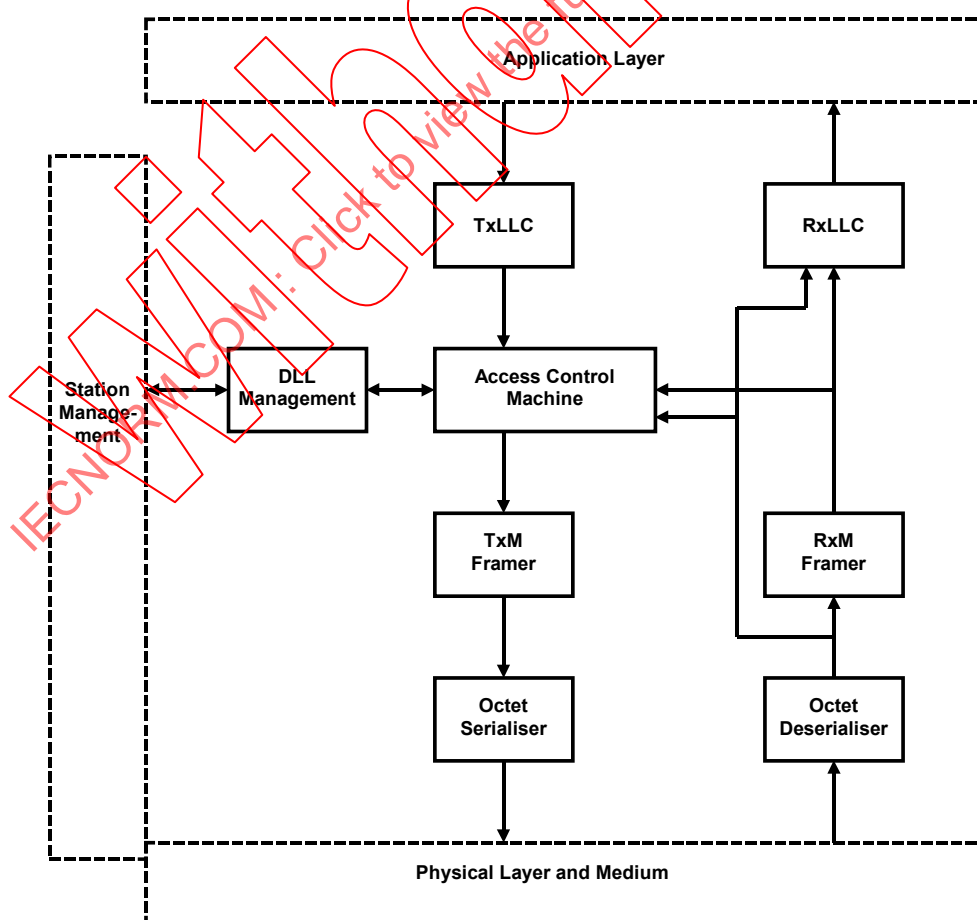
- de s'assurer que le nœud local détecte et utilise intégralement ses emplacements d'accès attribués dans la planification;
- de s'assurer que le nœud local ne gêne pas les transmissions des autres nœuds, particulièrement le nœud modérateur;
- de démarrer le NUT suivant (voir 4.1.2) planifié, que la DLPDU modérateur soit entendue ou pas;
- si le nœud local est le modérateur, de transmettre chaque DLPDU modérateur strictement selon la planification.

La couche de liaison de données contient les composants figurant dans le Tableau 1:

Tableau 1 – Composants de la couche de liaison de données

Composants	Description
Machine de contrôle d'accès (ACM)	reçoit et transmet les DLPDU de contrôle et les informations d'en-tête, et détermine le déroulement et la durée des transmissions.
Transmit LLC (TxLLC)	place en mémoire tampon les DLSDU reçues du gestionnaire de station et de l'utilisateur DLS, et détermine celle qu'il convient de transmettre ensuite.
Receive LLC (RxLLC)	procède à la mise en quarantaine des unités de liaison reçues tant qu'elles n'ont pas été validées par un FCS correct.
Transmit Machine (TxM)	reçoit les demandes à envoyer aux en-têtes et amorces de fin de bande du DLPDU et les Lpackets de l'ACM, et les découpe en demandes de symbole d'octet qui sont envoyés au convertisseur parallèle-série.
Receive Machine (RxM)	assemble les Lpackets reçus des symboles d'octet provenant du convertisseur série-parallèle et les soumet au RxLLC.
Convertisseur parallèle-série	reçoit les symboles d'octet, les code et les convertit, puis les envoie en tant que M_symbols à la PhL. Il est également chargé de générer le FCS.
Convertisseur série-parallèle	reçoit les M_symbols de la PhL, les convertit en octets et les envoie à la machine destinataire. Il est également chargé de vérifier le FCS.
Interface de gestion DLL	conserve les variables de gestion de station appartenant à la DLL, et permet de gérer les modifications synchronisées apportées aux paramètres de liaison.

La disposition interne de ces composants, et leurs interfaces, sont présentées dans la Figure 2. Les pointes de flèche illustrent la direction principale du flux de données et de contrôle.



Légende

Anglais	Français
application layer	couche d'application
station management	gestion de station
DLL management	gestion DLL
access control machine	machine de contrôle d'accès
TxM framer	structure Txm
RxM framer	structure RxM
octet serialiser	convertisseur parallèle-série d'octet
octet deserialiser	convertisseur série-parallèle d'octet
physical layer and medium	couche physique et support

Figure 2 – Architecture interne de la couche de liaison de données

4.1.2 Machine de contrôle d'accès (ACM) et fonctions de support de planification

Les fonctions ACM planifient toute la communication entre deux DLE. Le déroulement de cette communication est réglé sur deux niveaux,

- 1) pour fournir une distribution temporelle précise des opportunités planifiées des communications conçues conformément à la liaison locale et à un réseau étendu de liaisons, et avec une planification préalablement établie.
- 2) pour assurer une distribution démocratique des opportunités non planifiées des communications arbitraires, en général de manière cycle, mais asynchrone.

Le déroulement précis de la planification, tel qu'indiqué en 1), est essentiel à la prise en charge de nombreuses tâches de contrôle et de collecte des données dans le domaine des applications de ce protocole. La planification repose sur un cycle de durée fixe et répétitif appelé durée de mise à jour du réseau (NUT). La NUT est maintenue dans un synchronisme proche de tous les nœuds participants de la DLL et est utilisée pour fournir deux types d'opportunité d'accès;

- 1) Une opportunité pour chaque station active afin de transférer une DLPDU planifiée contenant plusieurs DLSDU planifiées.

NOTE 1 Seules les DLSDU planifiées comme leurs paramètres QoS peuvent être incluses dans une DLPDU planifiée.

NOTE 2 Les DLSDU planifiées comme leur paramètre QoS sont en général des transferts en mode connexion comme leur paramètre de balise.

- 2) Une opportunité, pour au moins une station active, d'envoyer une DLPDU (non planifiée) d'arrière-plan contenant plusieurs DLSDU de type QoS. Si le temps est disponible dans la NUT, une opportunité MAC d'arrière-plan est offerte à chaque station active pour leur adresse MAC ID. Le MAC ID de la station de la première opportunité d'arrière-plan de chaque NUT est incrémenté (modulo +1 pour l'ensemble d'adresses d'arrière-plan) pour chaque NUT successive afin de partager démocratiquement le temps disponible entre les stations actives.

Les modes opératoires de prise en charge sont inclus pour le transfert automatique vers les services de planification de sauvegarde et pour gérer l'utilisation du support Ph redondant.

4.1.3 Transfert de données en mode connexion, en mode sans connexion et service DL

Le mode connexion, le mode sans connexion et les services DL offrent les fonctionnalités nécessaires à la:

- gestion des interactions entre le fournisseur DL et l'utilisateur DL en convertissant les primitives de demande et de réponse en opérations DLE nécessaires, et en générant des primitives d'indications et de confirmation, le cas échéant,

- détermination de l'adresse DL et des informations de commande à ajouter à chaque DLSDU,
- confirmation locale de l'état de chaque DLSDU sortante,
- formation de DLPDU par concaténation de plusieurs DLSDU d'utilisateur DL pour le transfert efficace sous la forme de PhPDU uniques, soumis à des limitations de taille de DLPDU, aux paramètres QoS et au type d'opportunité d'accès.

4.2 Services fournis par la DL

4.2.1 Présentation

La DLL offre des services de transfert de données sans connexion et en mode connexion pour les DLSDU de taille limitée, un service synchronisé en interne, des services de planification du contrôle de l'attribution du temps du service PhL partagé sous-jacent, un service de gestion d'adresse DL(SAP) et de file d'attente, et un service de compression/décompression combinant plusieurs DLSDU en une seule DLPDU de transfert de données pour une utilisation efficace de chaque opportunité d'accès.

4.2.2 QoS

4.2.2.1 Définition

Le paramètre QoS spécifie les options de priorité des DLSDU et les opportunités d'accès. Les valeurs disponibles sont les suivantes:

4.2.2.2 Priorité planifiée par DLL

Le QoS de priorité planifiée assure l'envoi cyclique et acyclique temporel des DLSDU. Le déroulement d'exécution de ce service planifié de bus de terrain peut être précis et répété à 1 ms près. En général, la priorité planifiée est utilisée avec les services en mode connecté.

NOTE Le maître Keeper est chargé de la publication régulière d'un identifiant unique de table (TUI) permettant à chaque nœud de détenir une référence pour l'ensemble en cours de paramètres de fonctionnement de liaison. Cette publication du TUI utilise une balise fixe, et est envoyée avec l'ensemble de priorité QoS à la planification.

4.2.2.3 Priorité élevée DLL (non planifiée)

Le QoS de priorité élevée assure l'envoi acyclique des DLSDU avec un temps supérieur limité pour le délai d'envoi. Les données relatives à cette priorité sont envoyées uniquement si toutes les données planifiées ont été envoyées et qu'une opportunité d'envoi non planifiée est disponible.

4.2.2.4 Priorité basse DLL (non planifiée)

Le QoS de priorité basse assure l'envoi des DLSDU uniquement en fonction des disponibilités. Les données relatives à cette priorité sont envoyées uniquement si toutes les autres priorités de données ont été envoyées et qu'une opportunité d'envoi non planifiée est disponible.

NOTE Les priorités élevée et basse sont uniquement utilisées dans un sens local pour définir l'ordre de service des données utilisateur DLS soumises non planifiées.

4.3 Structure et définition des adresses DL

4.3.1 Généralités

Les adresses DL sont utilisées comme des adresses MAC ID (indicateurs de nœud de liaison), des adresses de balise fixe (adresses DLSAP et adresses DL de groupe) et des adresses de balise générique (adresses DLCEP).

Ce paragraphe définit la forme et la structure des adresses DL et inclus des définitions spécifiques pour certaines adresses DL standard.

Toutes les adresses sont composées d'octets. Chaque octet doit être en premier lieu transféré au bit inférieur de la couche Ph et plusieurs octets doivent être transférés en commençant par l'octet de poids faible (format petit-boutiste).

Trois types d'adresses DL sont utilisés.

4.3.2 Adresse MAC ID

Les adresses MAC ID identifient les nœuds physiques sur la liaison locale (voir la Figure 3).

Les valeurs admises sont comprises entre 0 et 99, les valeurs comprises entre 100 et 254 étant réservées.

MAC ID
1 octet

Figure 3 – Structure de base d'une adresse MAC ID

Le Tableau 2 spécifie l'utilisation des adresses MAC ID.

Tableau 2 – Allocation d'adresses MAC ID

MAC ID	Signification et objectif
0x0	adresse temporaire utilisée pour la maintenance
SMAX	valeur maximale en cours de l'adresse pour les opportunités d'accès planifiées. $SMAX \leq UMAX$
UMAX	valeur maximale en cours de l'adresse pour les opportunités d'accès non planifiées. $SMAX \leq UMAX \leq 0x63$
0x64 à 0xFF	réservée
0xFF	adresse de diffusion à tous les nœuds MAC ID de cette liaison

4.3.3 Adresse de balise générique

Les types d'adresse de balise générique identifient tous les transferts de données des services en mode connecté. La capacité disponible dans la DLL est de 3 octets (voir la Figure 4).

Balise générique
3 octets

Figure 4 – Structure de base d'une adresse de balise générique

NOTE 1 Les balises génériques permettent d'envoyer des DLSDU connectées et peuvent être associées à l'une des priorités QoS disponibles (planifiée, élevée, basse).

NOTE 2 La négociation et la gestion des balises génériques et des services en mode connecté relèvent de la responsabilité de l'utilisateur DL.

4.3.4 Adresse de balise fixe

Les types d'adresse de balise fixe identifient les DLSAP à l'intérieur du MAC ID de la station désignée. Ils sont utilisés avec tous les transferts de données pour les services en mode sans connexion.

NOTE Les balises fixes permettent d'envoyer des DLSDU sans connexion et sont en principe associées à des priorités QoS non planifiées (élevées, basses). En général, les balises fixes sont utilisées pour gérer la station et établir des connexions.

Chaque adresse de balise fixe doit être composée de deux octets (voir la Figure 5).

Service de balise fixe	MAC ID de destination
1 octet	1 octet

Figure 5 – Structure de base d'une adresse de balise fixe

Le premier octet doit spécifier le point d'accès au service dans le nœud de destination (voir le Tableau 3). Le deuxième octet doit être l'adresse du nœud de destination. L'adresse de destination doit être un MAC ID ou 0xFF, indiquant l'adresse de diffusion de tous les MAC ID sur la liaison locale.

Tableau 3 – Définitions de service de balise fixe

Service de balise fixe	Signification
0x00	modérateur
0x01 - 0x08	spécifique au fournisseur
0x09	demande ping
0x0A - 0x14	spécifique au fournisseur
0x15	tMinus
0x16 - 0x28	spécifique au fournisseur
0x29	réponse ping
0x2A - 0x3F	spécifique au fournisseur
0x40 - 0x6F	réservé
0x70 - 0x7F	spécifique au fournisseur
0x80	I'm alive
0x81	paramètres de liaison
0x82	réservé
0x83	UCMM
0x84	TUI
0x85	IP (réservé au protocole Internet)
0x86	WAMI
0x87	réservé
0x88	Keeper UCMM
0x89	Ethernet (utilisé pour le protocole de résolution d'adresse)
0x8A - 0x8B	réservé
0x8C	Répartition temporelle
0x8D - 0x8F	réservé à la répartition temporelle
0x90	débogage
0x91 - 0xCF	réservé
0xD0 - 0xEF	adresses de groupe
0xF0 - 0xFF	spécifique au fournisseur

Les balises fixes comprises entre 0xD0 et 0xEF doivent être réservées afin de permettre le tramage de groupe des identifiants de connexion.

4.4 Service supposé provenir de PhL

4.4.1 Exigences générales

Les paragraphes suivants contiennent les exigences relatives à l'interface d'une PhL de Type 2 qui ne sont pas requises pour les autres types.

- Machine de transmission.
- Machine destinataire.

- Convertisseur parallèle-série.
- Convertisseur série-parallèle.

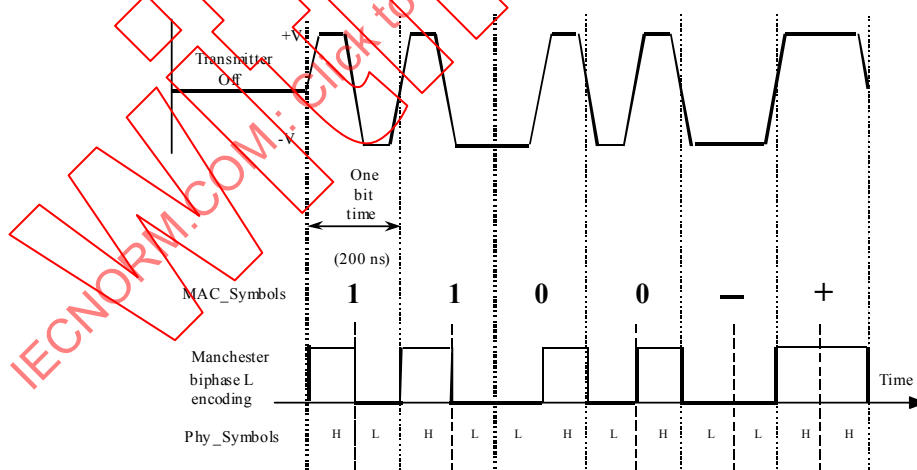
Cela est nécessaire car les unités de données d'interface Ph (PhIDU) que le Type 2 transmet par l'interface DLL-PhL sont des données en série individuelles et des symboles sans données, conçus du côté DLE en tant que M_Symbols (voir le Tableau 4), alors que les PhIDU des autres types ne sont pas conceptualisées de cette manière. En conséquence, une couche de liaison de données de Type 2 requiert une PhL de Type 2.

4.4.2 Règles de codage des données

Les M_Symbols transférés au niveau de l'interface DLL-PhL sont de quatre types, appelés 0, 1, +, -. Ils sont codés à l'intérieur de la PhL dans les Ph_Symbols appropriés (voir le Tableau 4). M_0 et M_1 sont codés en Ph_Symbols représentant les règles de codage biphasé L Manchester. Les symboles M_ND sont des symboles sans données afin de pouvoir créer des schémas de données uniques utilisés pour les délimiteurs de début et de fin. L'exemple de la forme d'onde de la tension de signal telle qu'elle peut être appliquée à un support conducteur est présenté sous une forme idéalisée dans la Figure 6 pour illustrer les règles de codage des données présentées dans le Tableau 4.

Tableau 4 – Règles de codage des données

Bits de données (nom commun)	Représentation M_Symbol	Codage Ph_Symbol	Codé Manchester
données « zéro »	M_0 ou {0}	{L,H}	0
Données « un »	M_1 ou {1}	{H,L}	1
« non_data+ »	M_ND+ ou {+}	{H,H}	pas de donnée
“non_data-”	M_ND- ou {-}	{L,L}	pas de donnée



Légende

Anglais	Français
transmitter off	émetteur désactivé
one bit time	un temps binaire
Manchester biphas L encoding	codage biphasé L Manchester
time	temps

Figure 6 – M_symbols et codage Manchester à 5 MHz (informative)

4.4.3 Interface DLL-PhL

4.4.3.1 Généralités

Il n'est pas utile d'exposer l'interface DLL-PhL dans la mise en œuvre d'une variante PhL. Cette interface peut être propre au nœud. Si la conformité à l'interface DLL-PhL est demandée, elle doit être conforme aux exigences du présent article.

4.4.3.2 Ph_lock_indication

ph_lock_indication doit fournir une indication du verrouillage de données ou de la synchronisation Ph_Symbol réalisée par PhL. Les états valides de ph_lock_indication doivent être « true » et « false ». La valeur doit être true en présence de Ph_Symbols valides dans l'interface PhL et lorsque la temporisation de l'interface PhL-DLL de M_Symbols est conforme aux exigences de PhL. Elle doit être « false » entre les trames (lorsque le support ne contient pas de Ph_Symbols), en cas de perte de la synchronisation des données ou lorsque la temporisation n'est pas conforme aux exigences de PhL. La valeur de ph_lock_indication doit « true » avant de lancer le délimiteur de début.

4.4.3.3 Ph_frame_indication

ph_frame_indication doit fournir une indication d'une trame de données valide provenant de PhL. Les valeurs d'état valides de ph_frame_indication doivent être « true » et « false ». La valeur de ph_frame_indication doit être « true » lorsque ph_lock_indication = true et lors de la réception du premier délimiteur de début valide. La valeur de ph_frame_indication doit être « false » à la réception du symbole M_ND suivant (après le délimiteur de début) ou lorsque ph_lock_indication = false.

NOTE Ce signal fournit une synchronisation d'octet à la DLL.

4.4.3.4 Ph_carrier_indication

ph_carrier_indication doit indiquer la présence d'une porteuse de signal sur le support. La valeur de ph_carrier_indication doit être « true » si la valeur d'identification de PhL interne de la porteuse de signal était « true » au cours de l'un des 4 dernières temporisations de M_symbol. Sa valeur doit être « false » dans le cas contraire.

4.4.3.5 Ph_data_indication

ph_data_indication doit représenter les M_Symbols décodés à partir des représentations internes de PhL (voir le Tableau 4, par exemple). Les M_symbols valides doivent être M_0 {0}, M_1 {1}, M_ND+ {+} et M_ND- {-} (voir le Tableau 5).

Tableau 5 – Symboles de données M

Bits de données (nom commun)	Représentation M_Symbol
données « zéro »	M_0 ou {0}
Données « un »	M_1 ou {1}
« non_data+ »	M_ND+ ou {+}
“non_data-”	M_ND- ou {-}

ph_data_indication doit représenter les M_Symbols décodés dans la PhL à chaque fois que la valeur de ph_lock_indication est « true ».

4.4.3.6 Ph_status_indication

ph_status_indication doit représenter l'état du délimiteur de la trame reçue de PhL (voir le Tableau 6). Les symboles valides doivent être Normal, Abort et Invalid. ph_status_indication

doit indiquer Normal suite à la réception d'une trame (ph_frame_indication = true) composée d'un délimiteur de début, de données valides (pas de symbole M_ND) et d'un délimiteur de fin. ph_status_indication doit indiquer Abort suite à la réception d'une trame (ph_frame_indication = true) composée d'un délimiteur de début, de données valides et d'un deuxième délimiteur de début. ph_status_indication doit indiquer Invalid suite à la réception d'une trame (ph_frame_indication = true) composée d'un délimiteur de début et de la détection d'un symbole M_ND ne faisant pas partie d'un délimiteur de début ou de fin.

Tableau 6 – Tableau de vérité de ph_status_indication

Ph_status_indication	Ph_frame_indication	Délimiteurs de début d'une seule trame	Détection de délimiteur de fin	Violations de code Manchester sans délimiteur
Normal	true	1	true	false
Abort	true	2	inutile	false
Invalid	true	1	inutile	true

4.4.3.7 Ph_data_request

ph_data_request doit présenter le M_Symbols à transmettre. Les symboles valides doivent être M_0, M_1, M_ND+ ou M_ND- (voir le Tableau 5). ph_data_request doit indiquer M_0 si aucune donnée ne doit être transmise (et que ph_frame_request = false).

4.4.3.8 Ph_frame_request

La valeur de ph_frame_request doit être « true » si ph_data_request présente les M_Symbols à coder dans les Ph_Symbols appropriés par la PhL. Elle doit être « false » si aucun M_Symbols valide ne doit être transféré vers PhL.

4.5 Classes fonctionnelles

Les mises en œuvre de ce protocole appartiennent à deux grandes classes:

- auteurs de la connexion (parfois appelés lecteurs laser);
- répondeurs de connexion ou cibles de connexion (parfois appelés Adaptateurs).

Les dispositifs dotés de fonctions d'émission de connexion contiennent des fonctions telles que les objets de planification, le support des outils de configuration externes et l'objet Keeper.

En général, les répondeurs de connexion sont des dispositifs plus simples qui n'ont pas besoin de toutes ces capacités.

Cette spécification ne préconise pas un groupement particulier de fonctions pour les classes de dispositifs de mise en œuvre.

NOTE Le groupement de services et de fonctions en classes de mise en œuvre peut être assuré par des spécifications de profils distinctes.

5 Structure générale et codage des PhIDU et DLPDU et éléments de procédure connexes

5.1 Présentation

La DLL et ses modes opératoires sont nécessaires pour fournir les services proposés à l'utilisateur DLS à l'aide des services disponibles auprès de PhL. Les principaux services de ce protocole sont:

- a) disposition d'opportunités d'accès opportun aux services DL de transfert de données planifiés du support sur un lien et sur un réseau étendu à liaisons multiples;
- b) la compression de plusieurs transferts d'utilisateur DLS en une seule unité de transfert PhL (DLPDU) afin d'utiliser efficacement chaque opportunité de transfert;
- c) la répartition efficace des données d'utilisateur DLS entre un utilisateur DLS de publication et un ensemble d'utilisateurs DLS d'abonnement;
- d) la disposition d'un sens synchronisé d'horloge interne parmi les DLE, disponible aux utilisateurs DLS de la liaison étendue;
- e) la disponibilité normalisée des capacités de gestion de l'adresse DL locale, de la mémoire tampon et de la file d'attente.

5.2 Mode opératoire d'accès au support

La couche de liaison de données a pour principal objet de déterminer, conjointement avec d'autres couches de la même liaison, les autorisations de transmission sur le support. Au niveau de son interface, la DLL offre des services de réception et de fourniture d'unités de données pour l'utilisateur DLS et la gestion de station.

Le protocole DLL repose sur un cycle de durée fixe et répétitif appelé durée de mise à jour du réseau (NUT). La NUT est maintenue dans un synchronisme proche de tous les nœuds de la liaison. Un nœud n'est pas autorisé à transmettre sur le support si la NUT n'en convient pas avec celle utilisée au niveau d'une liaison. Plusieurs liaisons peuvent présenter des durées de mise à jour du réseau différentes.

Chaque nœud contient son temporisateur synchronisé avec la NUT de la liaison locale. L'accès au support est déterminé par une subdivision locale de la NUT en emplacements d'accès. L'accès au support s'effectue dans l'ordre séquentiel en fonction du MAC ID du nœud. Des comportements particuliers ont été intégrés dans le protocole d'accès, permettant à un nœud prenant provisoirement en compte un MAC ID nul de procéder à des opérations de maintenance sur la liaison. Les numéros MAC ID de tous les nœuds d'une liaison sont uniques. Une DLL qui détecte la présence d'un MAC ID en double doit immédiatement arrêter la transmission.

Un mécanisme de transmission de jeton implicite est utilisé pour autoriser l'accès au support. Chaque nœud surveille le MAC ID source de chaque DLPDU reçue. A la fin d'une DLPDU, chaque DLL attribue un « registre de jeton implicite » au MAC ID source reçu, implémenté de 1. Si le registre de jeton implicite est égal au MAC ID local, le nœud doit transmettre une DLPDU contenant un ou plusieurs Lpackets avec des données ou une DLPDU nulle. Dans tous les autres cas, le nœud recherche une autre DLPDU dans le nœud identifié par le « registre de jeton implicite » ou une valeur de délai d'attente, si le nœud identifié ne peut assurer la transmission. Dans chaque cas, le « jeton implicite » est automatiquement avancé au MAC ID suivant. Tous les nœuds portent la même valeur dans leur « registre de jeton implicite », empêchant les collisions sur le support.

Le délai d'attente (appelé « intervalle de temps ») est fonction de la durée requise pour que:

- le nœud en cours entende la fin de la transmission du nœud précédent;
- le nœud en cours commence à transmettre;
- le nœud suivant entende le début de la transmission du nœud précédent.

L'intervalle de temps est ajusté de manière à compenser la longueur totale du support, le délai de propagation du support ayant un impact sur le premier et le dernier élément de la liste précédente.

NOTE Le calcul de l'intervalle de temps est spécifié en 8.2.

Chaque NUT est divisée en trois parties principales: planifiée, non planifiée et bande de garde (voir la Figure 7). Cette séquence est répétée dans chaque NUT. Le mécanisme de

transmission de jeton implicite permet d'autoriser l'accès au support pendant les intervalles planifiés et non planifiés.

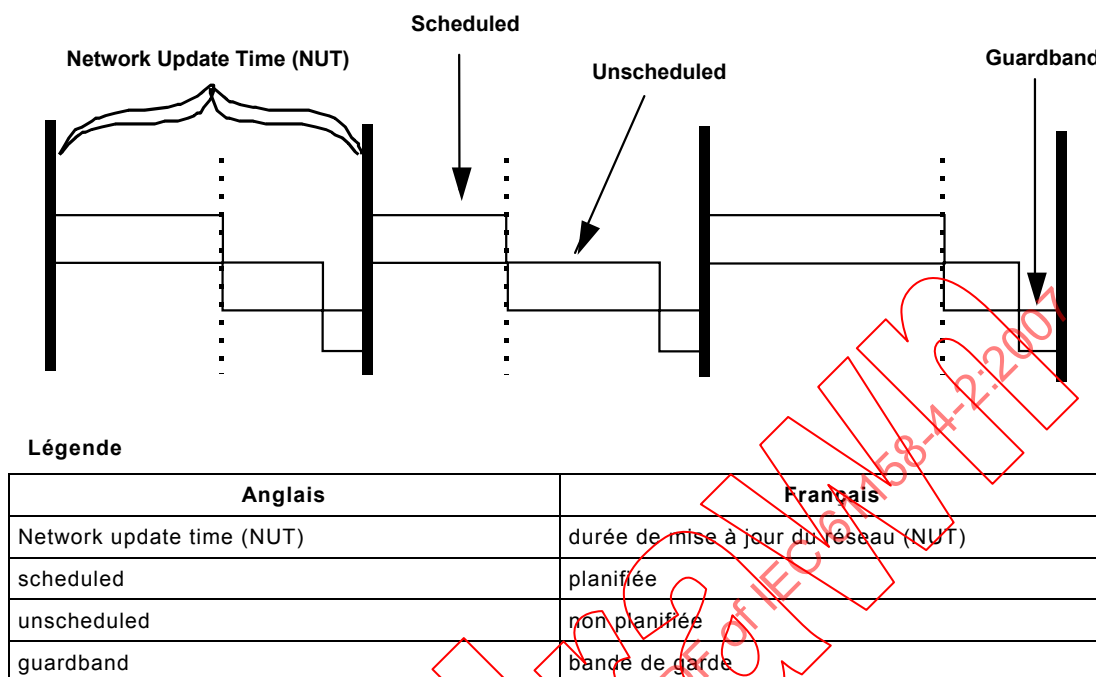
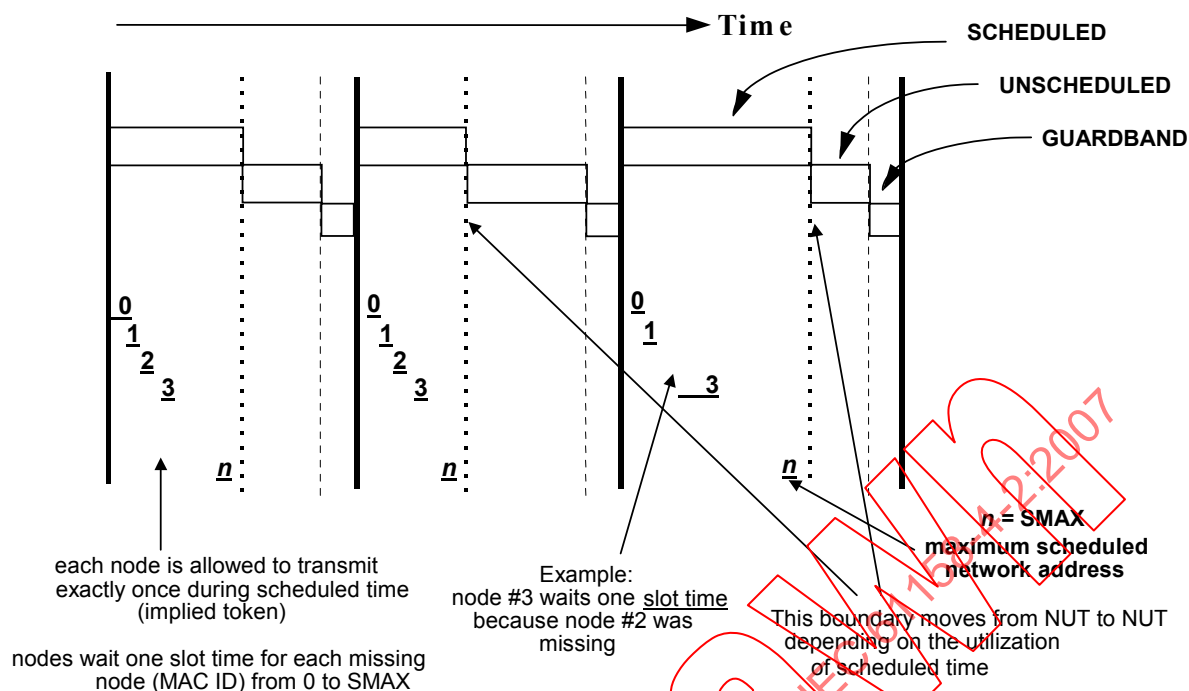


Figure 7 – Structure de la NUT

Au cours de la partie non planifiée de la NUT, chaque nœud (du nœud 0 au nœud SMAX) a une chance de transmettre des données (planifiées) prioritaires. SMAX est le MAC ID du nœud le plus élevé ayant accès au support pendant la partie planifiée de la NUT. Chaque nœud entre 0 et SMAX ne dispose que d'une seule opportunité d'envoi d'une DLPDU de données planifiées dans chaque NUT. L'opportunité d'accès au support pendant la durée planifiée est la même pour tous les nœuds de chaque NUT. Cela permet d'envoyer les données transmises pendant la partie planifiée de la NUT de manière prévisible et déterministe. La Figure 8 illustre l'autorisation de transmission pendant la durée planifiée. L'utilisateur DLS ajuste la quantité de données que chaque nœud peut transmettre au cours de la transmission planifiée de jeton.

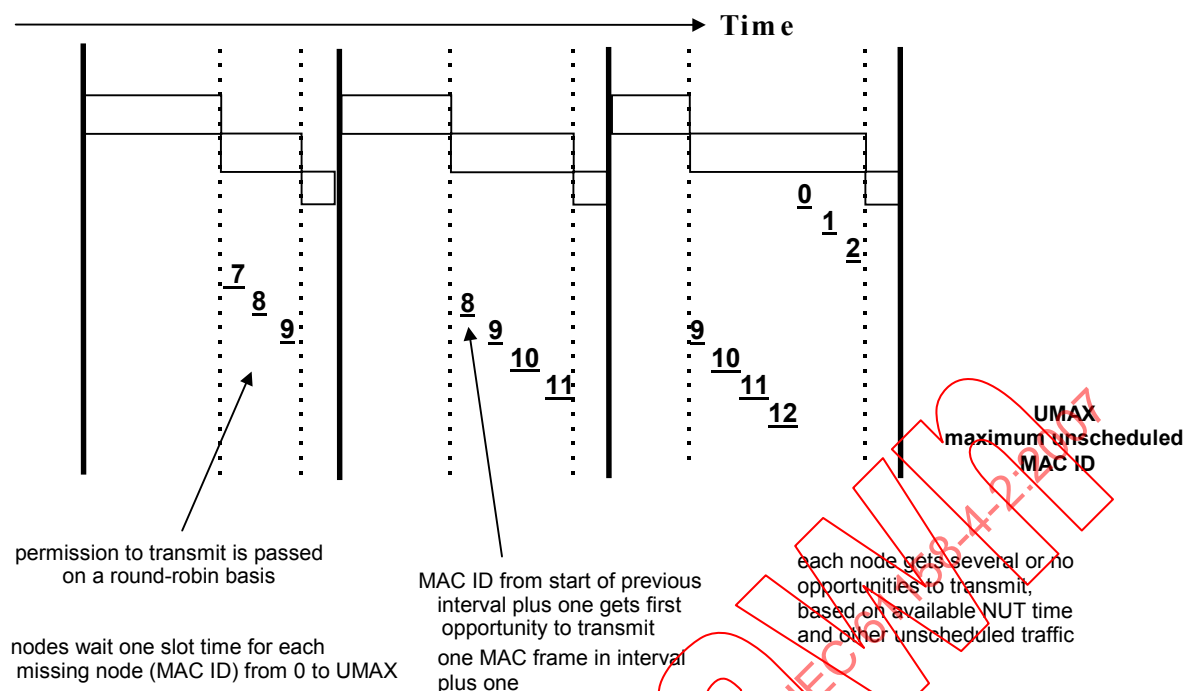


Légende

Anglais	Français
Time	durée
scheduled	planifiée
unscheduled	non planifiée
guardband	bande de garde
maximum scheduled network address	adresse de réseau planifiée maximum
each node ... (token)	chaque nœud est autorisé à transmettre exactement une fois pendant la durée planifiée (jeton impliqué)
example ... missing	exemple: le nœud 3 attend un intervalle de temps car le nœud 2 était manquant
this boundary ... time	cette limite se déplace d'une NUT à l'autre en fonction de l'utilisation de la durée planifiée
nodes wait ... SMAX	les nœuds attendent un intervalle de temps pour chaque nœud manquant (MAC ID) entre 0 et SMAX

Figure 8 – Accès au support pendant la durée planifiée

Au cours de la partie non planifiée de la NUT, tous les nœuds entre 0 et UMAX partagent l'opportunité de transmettre une DLPDU de données non critiques prioritaires de manière circulaire, tant que la durée NUT attribuée n'est pas épuisée. UMAX est le MAC ID du nœud le plus élevé ayant accès au support pendant la partie non planifiée de la NUT. La méthode circulaire d'opportunité d'accès permet à chaque nœud entre 0 et UMAX d'avoir zéro, une ou plusieurs opportunités d'envoyer des données non planifiées, selon la quantité de NUT restante à l'issue de la durée planifiée. Les variations du trafic planifié indiquent que l'opportunité d'accès au support pendant la durée non planifiée peut être différente pour chaque nœud d'une NUT. La Figure 9 illustre comment l'autorisation de transmission est accordée pendant la durée non planifiée. Le MAC ID du premier nœud de la partie non planifiée de la NUT est incrémenté de 1 pour chaque NUT. Le jeton non planifié commence par le MAC ID spécifié dans le registre de démarrage non planifié (USR) de la précédente DLPDU modérateur. L'USR incrémente chaque NUT de un modulo (UMAX+1). Si l'USR atteint UMAX avant la bande de garde, il revient à zéro et la transmission de jeton se poursuit.



Légende	
Anglais	Français
Time	durée
maximum scheduled MAC ID	MAC ID planifié maximum
permission ... basis	l'autorisation de transmettre est donnée de manière circulaire
MAC ID ... plus one	Le MAC ID du début de l'intervalle précédent incrémenté de un obtient la première opportunité de transmettre une trame MAC dans l'intervalle plus un
each node ... traffic	chaque nœud obtient plusieurs ou aucune autorisation de transmettre, sur la base de la durée NUT disponible et d'un autre trafic non planifié
nodes wait ... UMAX	les nœuds attendent un intervalle de temps pour chaque nœud manquant (MAC ID) entre 0 et UMAX

Figure 9 – Accès au support pendant la durée non planifiée

Lorsque la bande de garde est atteinte, tous les nœuds arrêtent de transmettre. Un nœud n'est pas autorisé à lancer une transmission qui risque de ne pas se terminer avant le début de la bande de garde. Au cours de la bande de garde, le nœud comportant le MAC ID le plus bas (appelé « modérateur ») transmet un message de maintenance (appelé « DLPDU modérateur ») qui:

- conserve les temporisateurs NUT de tous les nœuds synchronisés;
- publie les paramètres de liaison critiques permettant à tous les membres du groupe DLL local de partager une version commune des valeurs DLL importantes (NUT, intervalle de temps, SMAX, UMAX, par exemple).

Le modérateur transmet la DLPDU modérateur, qui resynchronise tous les nœuds et redémarre la NUT. Suite à la réception d'une DLPDU modérateur valide, chaque nœud compare ses valeurs internes à celles transmises dans la DLPDU modérateur. Un nœud utilisant des paramètres de liaison qui ne correspondent pas se désactive lui-même. Si la DLPDU modérateur n'est pas entendue pendant 2 NUT consécutives, le nœud comportant le MAC ID le plus bas joue le rôle de modérateur et commence à transmettre la DLPDU modérateur dans la bande de garde de la troisième NUT. Un nœud modérateur qui remarque la présence d'un autre nœud en ligne transmettant avec un MAC ID inférieur au sien annule immédiatement son rôle de modérateur.

Les situations classiques susceptibles d'engendrer une interruption du protocole d'accès DLL sont les suivantes:

- bruit induit sur la liaison;
- faible qualité du câble ou du raccordement;
- connexion physique de deux liaisons alors que le réseau est en cours de fonctionnement.

En général, ce type d'interruption a pour conséquence d'amener les nœuds à contester le nœud qu'il convient de transmettre. Il s'agit alors d'une « non-occurrence » de réseau. Un autre problème se produit lorsque les nœuds n'admettent pas les mêmes paramètres de configuration de liaison. Un nœud qui conteste les paramètres de liaison transmis par le modérateur est dit « suspect » et arrête immédiatement la transmission. Le protocole d'accès DLL est conçu pour récupérer un nœud suspect et le ramener en ligne.

5.3 Structure et codage de DLPDU

5.3.1 Généralités

Les DLPDU sont des DLPDU transmises sur la couche Ph pour l'envoi sur le support Ph. Les DLPDU sont formées par le protocole TxLLC (contrôle de liaison logique de transmission) en deux étapes:

- Les DLSDU sont formées en unités de liaison (Lpackets);
- Plusieurs Lpackets sont compressés dans une DLPDU faisant l'objet d'un ensemble de contraintes liées au QoS requis, au type d'opportunité d'accès disponible et aux limites de longueur globale de DLPDU.

5.3.2 Composants DLPDU

La structure générale de la DLPDU est présentée dans la Figure 10 et gérée par les unités de support DLL TxFramer, RxFramer, Octet Serializer et Octet Deserializer (voir la Figure 2).

Les composants de la DLPDU doivent être transmis dans l'ordre suivant: préambule, délimiteur de début, MAC ID source, zéro ou plusieurs Lpackets, FCS et délimiteur de fin.

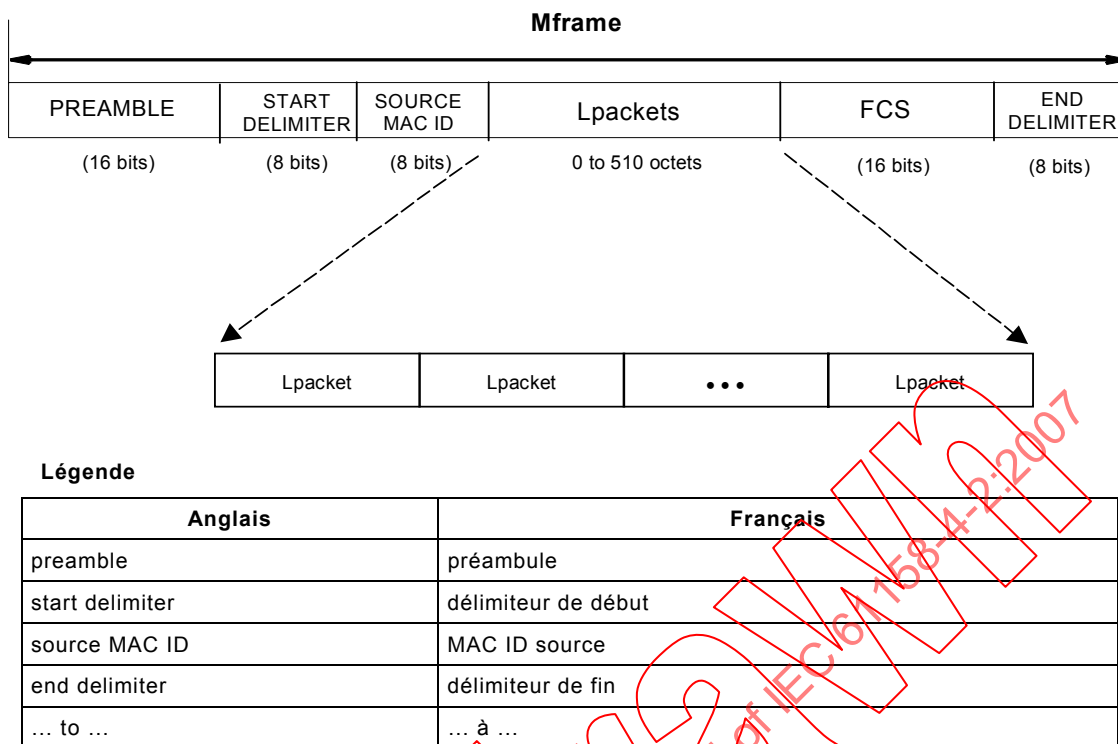


Figure 10 – Format de DLPDU

5.3.3 Préambule

Le préambule doit être composé de 16 {1} M_Symbols consécutifs.

5.3.4 Délimiteurs de début et de fin

Le délimiteur de début doit être composé des M_Symbols {+, 0, –, +, –, 1, 0, 1} transmis de gauche à droite.

Le délimiteur de fin doit être composé des M_Symbols {1, 0, 0, 1, +, –, +, –} transmis de gauche à droite.

L'utilisation des M_Symbols sans donnée doit être réservée pour les délimiteurs de début et de fin.

5.3.5 Octets DLPDU et ordre

Le MAC ID source, les Lpackets et la FCS doivent être composés d'octets. Un octet doit être composé d'exactly huit M_Symbols, chacun d'eux doit être {0} ou {1}. Un octet doit être d'abord transféré vers le bit inférieur PhL. Les champs multi-octets doivent être transmis, l'octet de poids faible l'étant en premier (format petit-boutiste).

5.3.6 MAC ID source

L'adresse du MAC ID source doit être composée d'un seul octet (voir 4.3.2). Un nœud peut également considérer provisoirement un MAC ID nul pour procéder à une maintenance de liaison.

5.3.7 Champ Lpackets

Aucun ou plusieurs Lpackets doivent être concaténés en une seule DLPDU faisant l'objet d'une contrainte liée au nombre maximal de 510 octets composant tous les Lpackets d'une DLPDU.

Une contrainte supplémentaire impose que les DLPDU envoyées au cours d'une opportunité d'accès planifiée ne doit contenir que des Lpackets avec un paramètre QoS planifié.

5.3.8 Séquence de contrôle de trame (FCS)

5.3.8.1 Exigences de séquence de contrôle de trame

La FCS doit être une somme de contrôle CCITT modifiée utilisant le polynôme de 16 bits: $G(X) = X^{16} + X^{12} + X^5 + 1$. Une FCS de deux octets doit être incluse dans chaque DLPDU.

Les concepts formels de génération et de contrôle à la réception sont présentés en 5.3.8.2. La génération et le contrôle de la FCS sont présentés plus en détails en 9.7 et 9.8.

5.3.8.2 Concepts formels de séquence de contrôle de trame

5.3.8.2.1 Champ de séquence de contrôle de trame

Dans le présent paragraphe, toute référence au bit K d'un octet fait référence au bit dont le poids dans un entier non signé d'un octet est 2^K .

NOTE Parfois appelé numérotation de bit « petit-boutiste ».

Dans le cadre de la présente norme, comme dans les autres Normes internationales (l'ISO/CEI 3309, l'ISO/CEI 8802 et l'ISO/CEI 9314-2, par exemple), la détection d'erreur au niveau de la DLPDU est assurée en calculant et ajoutant une séquence de contrôle de trame (FCS) à plusieurs bits aux autres champs DLPDU pendant la transmission. Il s'agit de former un « mot de code systématique »³ de longueur n , composé de k bits de message DLPDU suivis de $n - k$ (égal à 16) bits redondants, puis de déterminer pendant la réception que le message et la FCS concaténés forment un mot de code (n,k) légal. Le mécanisme de contrôle est présenté ci-dessous, la forme générique du polynôme générateur de cette construction FCS et le polynôme des données résiduelles du récepteur étant respectivement spécifiés dans les équations (4) et (9). Les polynômes spécifiques de la présente norme sont spécifiés dans le Tableau 7.

Tableau 7 – Longueur, polynômes et constantes FCS

Élément	Valeur
$n-k$	16
$G(X)$	$X^{16} + X^{12} + X^5 + 1$ (notes 1)
$R(X)$	$X^{12} + X^{11} + X^{10} + X^8 + X^3 + X^2 + X + 1$ (notes 2)
NOTE 1 Les mots de code $D(X)$ construits à partir de ce polynôme $G(X)$ comportent une distance 4 de Hamming pour les longueurs $\leq 4\ 095$ octets, et toutes les erreurs de poids paires sont détectées.	
NOTE 2 Il s'agit des mêmes polynômes et méthodes que ceux spécifiés dans l'ISO/CEI 3309 (HDLC).	
NOTE 3 Il convient que le $R(x)$ restant soit 0001 1101 0000 1111 (X^{15} à X^0 , respectivement) en l'absence d'erreur.	

5.3.8.2.2 Au niveau du DLE émetteur

Le message d'origine (c'est-à-dire la DLPDU sans FCS), la FCS et le mot de code de message composite (la DLPDU et la FCS concaténées) doivent être considérés comme des vecteurs $M(X)$, $F(X)$ et $D(X)$, de dimension k , $n - k$ et n , respectivement, dans un champ d'extension sur $GF(2)$. Si les bits de message sont $m_1 \dots m_k$ et que les bits FCS sont $f_{n-k-1} \dots f_0$, où

$m_1 \dots m_8$ forment le premier octet envoyé,

³ W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes* (2nd edition), MIT Press, Cambridge, 1972.

$m_{8N-7} \dots m_{8N}$ forment le $n^{\text{ème}}$ octet envoyé,
 $f_7 \dots f_0$ forment le dernier octet envoyé, et
 m_1 est envoyé par le(s) premier(s) symbole(s) PhL du message et f_0 est envoyé par le(s) dernier(s) symbole(s) PhL du message (sans compter les informations de trame PhL),

NOTE 1 Cet ordre « tel que transmis » est critique en fonction des propriétés de détection d'erreur de la FCS.

alors le vecteur de message $M(X)$ doit être

$$M(X) = m_1X^{k-1} + m_2X^{k-2} + \dots + m_{k-1}X^1 + m_k \quad (1)$$

et le vecteur FCS $F(X)$ doit être

$$\begin{aligned} F(X) &= f_{n-k-1}X^{n-k-1} + \dots + f_0 \\ &= f_{15}X^{15} + \dots + f_0 \end{aligned} \quad (2)$$

Le vecteur composite $D(X)$, de la DLPDU complète, doit être construit comme la concaténation du message et les vecteurs FCS

$$\begin{aligned} D(X) &= M(X)X^{n-k} + F(X) \\ &= m_1X^{n-1} + m_2X^{n-2} + \dots + m_kX^{n-k} + f_{n-k-1}X^{n-k-1} + \dots + f_0 \\ &= m_1X^{n-1} + m_2X^{n-2} + \dots + m_kX^{16} + f_{15}X^{15} + \dots + f_0 \quad (\text{si } k = 15) \end{aligned} \quad (3)$$

La DLPDU présentée à la PhL doit être composée d'une séquence d'octets dans l'ordre indiqué.

Les bits de contrôle redondants $f_{n-k-1} \dots f_0$ de la FCS doivent être les coefficients du reste $F(X)$, après division par $G(X)$, de $L(X)(X^k + 1) + M(X)X^{n-k}$

où $G(X)$ est le polynôme générateur de degré $n-k$ des mots de code

$$G(X) = X^{n-k} + g_{n-k-1}X^{n-k-1} + \dots + 1 \quad (4)$$

et $L(X)$ est le polynôme de poids maximal (tous) de degré $n-k-1$

$$\begin{aligned} L(X) &= (X^{n-k} + 1) / (X + 1) = X^{n-k-1} + X^{n-k-2} + \dots + X + 1 \\ &= X^{15} + X^{14} + X^{13} + X^{12} + \dots + X^2 + X + 1 \quad (\text{si } k = 15) \end{aligned} \quad (5)$$

En d'autres termes,

$$F(X) = L(X)(X^k + 1) + M(X)X^{n-k} \quad (\text{module } G(X)) \quad (6)$$

NOTE 2 Les termes $L(X)$ sont inclus dans le calcul afin de détecter la troncature ou l'extension initiale ou terminale de message en ajoutant un facteur dépendant de la longueur à la FCS.

NOTE 3 Comme mise en œuvre classique lorsque $n-k = 16$, le reste initial de la division est prédéfini sur chacun d'eux. Le train de bits de message transmis est multiplié par X^{n-k} et divisé (modulo 2) par le polynôme générateur $G(X)$, spécifié dans l'équation (7). Les compléments du reste résultant sont transmis comme FCS de $(n-k)$ bits, avec le coefficient de X^{n-k-1} transmis en premier.

L'équation de $D(X)$ ne s'applique pas au protocole de Type 8, car ce protocole envoie le message et les bits de contrôle dans des DLPDU distinctes. L'équation de $F(X)$ telle que donnée ne s'applique pas au protocole de Type 8, car ce protocole ne complète pas les bits de contrôle avant la transmission. L'équation correspondant au $F(X)$ du protocole de Type 8 est

$$F_{\text{Type 8}}(X) = L(X)X^k + M(X)X^{n-k} \quad (\text{module } G(X)) \quad (7)$$

NOTE 4 Comme mise en œuvre classique du protocole de Type 8, le reste initial de la division est prédéfini sur chacun d'eux. Le train de bits de message transmis est multiplié par X^{n-k} et divisé (modulo 2) par le polynôme générateur $G(X)$, spécifié dans l'équation (7). Le reste obtenu sans complément de uns est transmis en tant que FCS à $(n-k)$ bits, le coefficient de X^0 étant transmis en premier.

5.3.8.2.3 Au niveau du DLE destinataire

NOTE 1 Le présent paragraphe ne s'applique pas au protocole de Type 8.

La séquence d'octets indiquée par PhE doit être concaténée dans la DLPDU et la FCS reçues, puis considérée comme un vecteur $V(X)$ de dimension u

$$V(X) = v_1X^{u-1} + v_2X^{u-2} + \dots + v_{u-1}X + v_u \quad (8)$$

NOTE 2 En raison d'erreurs, u peut être différent de n , la dimension du vecteur de code transmis.

Un reste $R(X)$ doit être calculé pour $V(X)$, la DLPDU et la FCS reçues, par une méthode analogue à celle utilisée par le DLE émetteur (voir 5.3.8.2.2) dans le calcul de $F(X)$

$$\begin{aligned} R(X) &= L(X) X^u + V(X) X^{n-k} \text{ (modulo } G(X)) \\ &= r_{n-k-1}X^{n-k-1} + \dots + r_0 \end{aligned} \quad (9)$$

Définir $E(X)$ comme le vecteur de code d'erreur des différences (modulo-2) supplémentaires entre le vecteur de code transmis $D(X)$ et le vecteur reçu $V(X)$ résultant d'erreurs rencontrées (dans le fournisseur PhS et dans les ponts) entre les DLE émetteurs et destinataires.

$$E(X) = D(X) + V(X) \quad (10)$$

Si aucune erreur ne s'est produite, de sorte que $E(X) = 0$, alors $R(X)$ est égal à un polynôme de reste d'une constante non nulle

$$R_{ok}(X) = L(X) X^{n-k} \text{ (modulo } G(X)) \quad (11)$$

dont la valeur est indépendante de $D(X)$. Malheureusement, $R(X)$ est aussi égal à $R_{ok}(X)$, lorsque $E(X)$ est un multiple exact non nul de $G(X)$, auquel cas des erreurs sont « indétectables ». Dans tous les autres cas, $R(X)$ n'est pas égal à $R_{ok}(X)$. Ces DLPDU sont erronées et doivent être écartées sans autre forme d'analyse.

NOTE 3 Comme mise en œuvre classique, le reste initial de la division est prédéfini sur chacun d'eux. Le train de bits reçu est multiplié par X^{n-k} et divisé (modulo 2) par le polynôme générateur $G(X)$, spécifié dans l'équation (8).

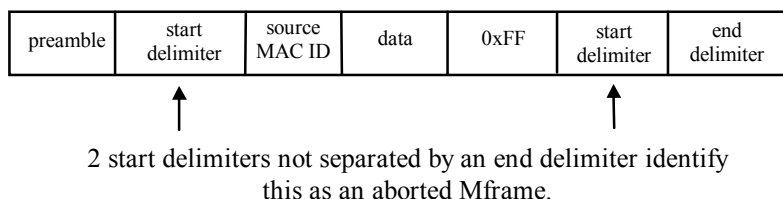
5.3.9 DLPDU nulle

Une DLPDU dont le champ L_{packet} est vide, appelée DLPDU nulle, doit être envoyée par un nœud pour indiquer que l'opportunité d'accès n'admet aucun envoi de données au niveau du QoS.

5.3.10 DLPDU annulée

Pour annuler une DLPDU partiellement transmise, la couche de liaison de données doit transmettre la séquence suivante:

- un octet 0xFF;
- le délimiteur de début;
- le délimiteur de fin.



Légende

Anglais	Français
preamble	préambule
start delimiter	délimiteur de début
source MAC ID	MAC ID source
data	données
end delimiter	délimiteur de fin
2 start ... Mframe	2 délimiteurs de début non séparés par un délimiteur de fin indiquent qu'il s'agit d'une Mframe annulée

Figure 11 – Annulation d'une DLPDU pendant la transmission

NOTE 1 Certaines mises en œuvre de la présente spécification peuvent générer la DLPDU en cours de transmission. Si une couche supérieure n'a fourni qu'une DLPDU partielle à la couche de liaison de données et que la DLL a commencé la transmission de la DLPDU en supposant que le reste va être bientôt fourni, un dépassement négatif de capacité de transmission se produit si le reste de la DLPDU n'est pas fourni à la DLL suffisamment tôt pour terminer la transmission.

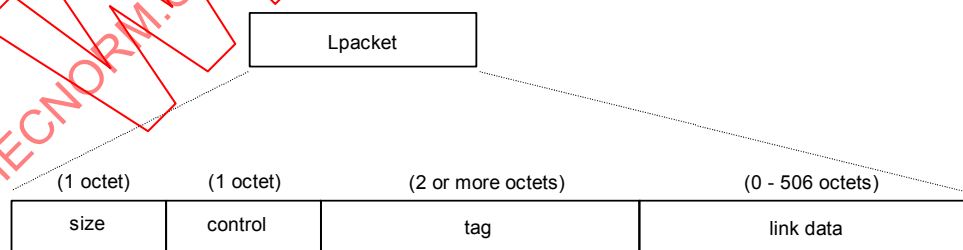
NOTE 2 Bien que la DLL assemble toute la DLPDU, y compris les délimiteurs, la PhL tente de détecter les délimiteurs de début et de fin. La PhL identifie une annulation de transmission comme une DLPDU contenant deux délimiteurs de début qui ne sont pas séparés par un délimiteur de fin (voir la Figure 11).

NOTE 3 L'octet 0xFF est inséré avant la paire délimiteur de début/délimiteur de fin afin d'éviter d'enfreindre des contraintes PhL régissant les relations de symbole physique consécutif sur le support.

5.4 Composants Lpacket

5.4.1 Structure générale de Lpacket

Lpacket (ou unité de liaison) est le PDU utilisé pour acheminer les DLSDU entre les utilisateurs DLS homologues. Les composants du Lpacket doivent être transmis dans l'ordre suivant: taille, contrôle, balise et données de liaison (voir Figure 12).



Légende

Anglais	Français
size	taille
control	contrôle
2 or more octets	2 octets ou plus
tag	balise
link data	données de liaison

Figure 12 – Format de Lpacket

5.4.2 Taille

Le champ de taille doit préciser le nombre de paires d'octets (entre 3 et 255) que contient un Lpacket individuel. Le nombre de paires d'octets doit inclure les champs de taille, de contrôle, de balise et de données de liaison dénombrés conformément aux règles suivantes:

- lorsqu'ils sont combinés, la taille et le contrôle doivent compter comme une seule paire d'octets;
- le nombre de paires d'octets des champs de balise et de données de liaison doit être déterminé séparément, et le nombre d'octets pairs doit être arrondi;
- que les paires d'octets soient comptées de manière indépendante, le format d'un Lpacket placé dans une DLPDU ne doit pas inclure les octets supplémentaires générés par l'arrondi.

NOTE Par exemple, le champ de taille d'un Lpacket doté d'une balise de 3 octets (2 paires d'octets) et de données de liaison de 9 octets (5 paires d'octets) est de $1 + 2 + 5 = 8$ paires d'octets. Toutefois, le Lpacket occupe $2 + 3 + 9 = 14$ octets dans la DLPDU.

5.4.3 Contrôle

5.4.3.1 Bit 0 et Bit 4 (type de Lpacket)

Les bits du champ de contrôle doivent être numérotés de 0 à 7. Le bit 0 doit être le bit de poids faible. Le bit 0 doit indiquer le type de Lpacket.

- Lorsqu'une valeur lui est attribuée (bit 0 = 1), il doit s'agir d'un Lpacket à balise fixe (voir 4.3.4).
- Lorsqu'il est supprimé (bit 0 = 0), il doit s'agir d'un Lpacket à balise générique (voir 4.3.3).

Le bit 4 du champ de contrôle doit être l'inverse du bit 0. Si le bit 0 est supprimé, une valeur doit être attribuée au bit 4. Si une valeur est attribuée au bit 0, le bit 4 doit être annulé.

5.4.3.2 Bit 1 (taille de balise impaire)

Le bit 1 du champ de contrôle doit indiquer si le champ de balise contient un nombre d'octets pair ou impair. S'il est annulé (bit 1 = 0), ce bit doit indiquer que la balise contient un nombre d'octets pair. Si une valeur lui est attribuée (bit 1 = 1), il doit indiquer que la balise contient un nombre d'octets impair.

5.4.3.3 Bit 2 (taille de données de liaison impaire)

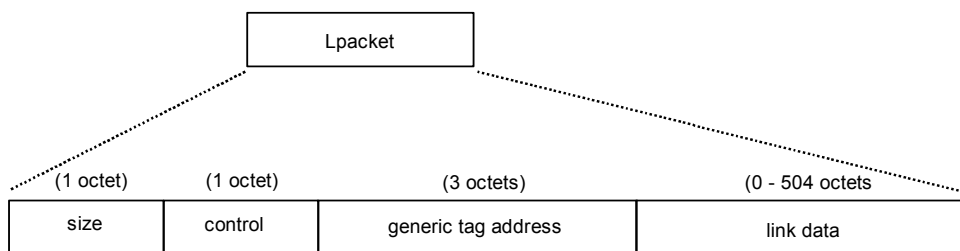
Le bit 2 du champ de contrôle doit indiquer si les données de liaison contiennent un nombre d'octets pair ou impair. S'il est annulé (bit 2 = 0), ce bit doit indiquer que les données de liaison contiennent un nombre d'octets pair. Si une valeur lui est attribuée (bit 2 = 1), il doit indiquer que les données de liaison contiennent un nombre d'octets impair.

5.4.3.4 Bits 3, 5, 6 et 7 (bits réservés)

Les bits 3, 5, 6 et 7 du champ de contrôle sont réservés et doivent être nuls.

5.4.4 Lpackets à balise générique

Les unités de liaison génériques sont formées à partir de la balise générique DLS et des données utilisateur DLS fournies par l'utilisateur DLS, en plus des informations des champs de taille et de contrôle relatives aux balises génériques (Figure 13). L'adresse de balise générique à 3 octets (voir 4.3.3) doit identifier les informations de mode connexion acheminées dans le champ de données de liaison.



Légende

Anglais	Français
size	taille
control	contrôle
generic tag address	adresse de balise générique
link data	données de liaison

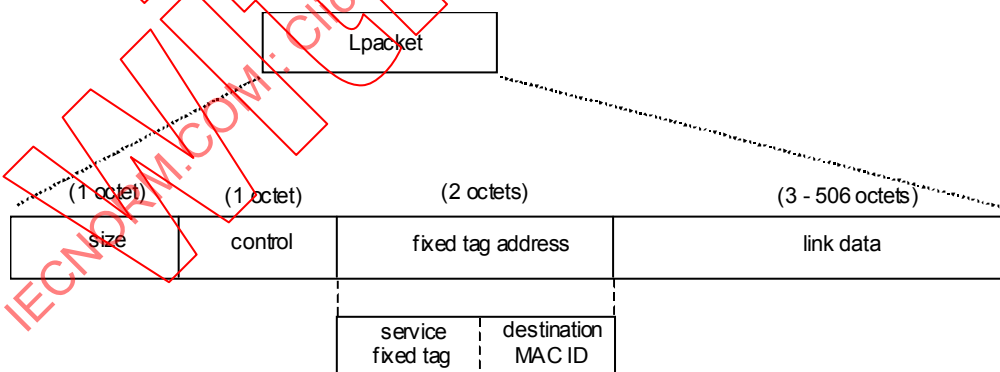
Figure 13 – Format Lpacket à balise générique

La longueur minimale du Lpacket à balise générique doit être de 5 octets (taille de données de liaison minimale de 0 octets). La taille maximale des données de liaison doit être de 504 octets.

NOTE En règle générale, les Lpackets génériques sont des transferts en mode connexion dont la valeur QoS est planifiée ou basse.

5.4.5 Lpackets à balise fixe

Les unités de liaison à balise fixe sont formées à partir de la balise fixe DLS et des données utilisateur DLS fournies par l'utilisateur DLS, en plus des informations des champs de taille et de contrôle relatives aux balises fixes (voir Figure 14). L'adresse de balise fixe à 2 octets (voir 4.3.4) doit identifier le point d'accès au service dans le nœud de destination et l'adresse de ce nœud.



Légende

Anglais	Français
size	taille
control	contrôle
fixed tag address	adresse de balise fixe
link data	données de liaison
service fixed tage	balise fixe de service
destination MAC ID	MAC ID de destination

Figure 14 – Format Lpacket à balise fixe

La longueur minimale du Lpacket à balise fixe doit être de 7 octets (taille de données de liaison minimale de 3 octets). La taille maximale des données de liaison doit être de 506 octets.

NOTE Les Lpackets fixes sont toujours des transferts en mode sans connexion. En général, ils ne sont pas envoyés avec la valeur du paramètre QoS planifiée.

5.5 Modes opératoires DLPDU

5.5.1 Généralités

Le protocole de transmission combine un ou plusieurs Lpackets en une seule DLPDU pour procéder à des envois en fonction de l'opportunité de transmission disponible. Les composants de DLPDU sont transmis à la couche Ph par TxM et le Convertisseur parallèle-série (voir 9.3).

Un nœud doit toujours envoyer une DLPDU à chaque opportunité d'accès valide pour qu'elle le transmette, et aucune DLPDU ne doit contenir des Lpackets avec un paramètre QoS inférieur à la valeur QoS applicable à l'opportunité d'accès.

Si un nœud ne contient pas de Lpackets dont la valeur est supérieure ou égale au QoS de l'opportunité d'accès, une DLPDU nulle doit être envoyée.

Si, pour des raisons internes, un nœud commence une DLPDU et n'est pas en mesure de la terminer, la séquence d'annulation doit être envoyée, la transmission doit s'arrêter pour cette opportunité d'accès et l'utilisateur DLS local doit en être informé.

NOTE Les DLPDU nulles et les DLPDU annulées informent les autres nœuds que l'utilisation de l'intervalle de transmission est terminée et autorisent le nœud suivant à commencer la transmission sans attendre le délai d'attente de l'intervalle.

Chaque Lpacket de la file d'attente de transfert comporte un identifiant local. A l'issue du transfert vers la PhL ou d'un transfert annulé, l'identifiant local est renvoyé à l'utilisateur DLS local avec une confirmation d'état OK ou TXABORT.

L'utilisateur DLS dispose de services locaux supplémentaires pour identifier les Lpackets non envoyés et demander leur éviction de la file d'attente, cette action renvoyant l'état de confirmation FLUSHED avec l'identifiant local des Lpackets évincés.

5.5.2 Envoi de DLPDU planifiées

En règle générale, les DLPDU planifiées permettent de transférer des opérations en mode connecté à l'aide de Lpackets génériques. Elles sont issues de la concaténation de plusieurs Lpackets (limités à 510 octets), commençant par des informations de contrôle Ph et un MAC ID de station source, et se terminant par une séquence de contrôle de trame et d'autres informations de contrôle Ph.

Les Lpackets sont compressés dans une DLPDU planifiée dans l'ordre FIFO, tels que l'utilisateur DLS les a envoyés, tant que la file d'attente contient encore des paquets avec un QoS planifié ou que la taille du Lpacket suivant ne dépasse pas l'espace restant.

5.5.3 Envoi de DLPDU non planifiées

Les DLPDU non planifiées peuvent contenir des Lpackets génériques et des Lpackets fixes. Elles sont issues de la concaténation de plusieurs Lpackets (limités à 510 octets), commençant par des informations de contrôle Ph et un MAC ID de station source, et se terminant par une séquence de contrôle de trame et d'autres informations de contrôle Ph.

Les Lpackets sont compressés dans une DLPDU non planifiée dans l'ordre de priorité QoS (priorités Planifiée, Élevée et Basse, respectivement). Dans chaque QoS, l'ordre FIFO de

mise en file d'attente par l'utilisateur DLS est respecté, tant que la file d'attente n'est pas vide ou que la taille du Lpacket suivant ne dépasse pas l'espace restant.

5.5.4 DLPDU destinataires

5.5.4.1 Décompression

Les DLPDU reçues sont assemblées à partir du flux de symboles Ph entrant et décompressées par les fonctions Octet Deserializer, Receive Machine Framer (RxM) et Receive Logical Link Control (RxLLC) en deux étapes:

- une séquence de contrôle de trame est calculée sur le flux d'octets reçus (à l'exclusion des délimiteurs Ph) et la valeur résiduelle est vérifiée, si elle est correcte, puis
- chaque Lpacket que contient la DLPDU est examiné pour extraire sa balise. Chaque balise extraite est ensuite comparée au contenu d'un filtre de balise local pour déterminer si le Lpacket associé est pertinent pour la station destinataire. Les Lpackets dont les balises correspondent au filtre de balise font l'objet d'un traitement supplémentaire, puis sont transmis à l'utilisateur DL, au gestionnaire DL ou utilisés dans la DLL. Les indications transmises à l'utilisateur DL ou au gestionnaire DL contiennent la taille, la balise et les données du Lpacket. En outre, pour les Lpackets à balise fixe, l'ID source est également indiqué.

5.5.4.2 Filtrage de balise

Chaque fournisseur DLS local gère une liste de balises permettant d'identifier les Lpackets pertinents pour son utilisateur DL ou ses fonctions de gestion DL internes. L'utilisateur DL peut accéder à cette liste et la modifier pour spécifier:

- les balises génériques intéressantes;
- les balises fixes intéressantes.

NOTE Une balise générique spécifie les Lpackets d'un seul objet en mode connexion. Une balise fixe spécifie tous les messages en mode sans connexion d'une classe particulière (Répartition temporelle, par exemple).

5.6 Récapitulatif des services de support et objets DLL

Le fonctionnement et la coordination des fournisseurs DLL participants sont déterminés par des variables et des attributs placés dans des objets de gestion DL ou fournis par les services utilisateur DL figurant dans le Tableau 8. Ces éléments sont accessibles par l'intermédiaire d'adresses de balise fixe définies, de services utilisateur DL ou de services locaux non spécifiés. Les détails sont présentés dans les articles figurant dans la liste.

Tableau 8 – Services de support et objets DLL

Élément DLL	Description	Emplacement
Services utilisateur DLS		Voir CEI 61158-3-2
Service de transfert en mode connecté	service local de demande, d'indication et de confirmation de transfert de données de balise générique (mode connecté) par les Lpackets	Voir 6.2.2
Service de transfert en mode sans connexion	service local de demande, d'indication et de confirmation de transfert de données de balise générique (mode connecté) par les Lpackets	Voir 6.2.2
Service de maintenance de la file d'attente	Service local d'éviction des messages Lpacket non envoyés de la file d'attente de service DL interne	Voir 6.2.3
Service de filtrage de balise	Service local destiné à indiquer au fournisseur DL local de l'utilisateur DLS les balises Lpacket et les types de balise que la DLL doit accepter et traiter ou transmettre à l'utilisateur DLL	Voir 6.2.4

Élément DLL	Description	Emplacement
Service de synchronisation de liaison	Service local destiné à informer l'utilisateur DLS local du numéro NUT en cours pour maintenir les pointeurs planifiés	Voir 6.2.5
Service de modification de paramètre synchronisé et compte à rebours de début tMinus	Service local de chargement et de modification des listes locales de données de configuration DL en cours et en attente et, conjointement au service de compte à rebours de début tMinus et à la DLPDU modérateur, de gestion d'une modification précisément synchronisée aux nouvelles configurations de liaison DL	Voir 6.2.6
Service de rapport d'événements	Services locaux rapportant les différents événements utilisés par le gestionnaire	Voir 6.2.7
Service de FCS erroné		Voir 6.2.8
Modérateur en cours		Voir 6.2.9
Mise sous tension et mise en ligne		Voir 6.2.10
Activation du modérateur		Voir 6.2.11
Écoute uniquement		Voir 6.2.12
Service de balise générique	Service Lpacket pour la communication en mode connecté	Voir 6.3
Services de balise fixe	Ensemble d'adresses de service Lpacket pour différentes communications en mode sans connexion	Voir 5.4.5
modérateur	Diffusion de Lpacket dans chaque NUT pour la gestion de la précision temporelle et la synchronisation de modification des paramètres de liaison	Voir 6.4
Service de répartition temporelle	Lpacket de publications des valeurs et décalages temporels utilisés dans la coordination du même sens de la durée dans toutes les stations	Voir 6.5
UCMM	Service Lpacket de transfert des DLSDU entre un utilisateur DLS et l'UCMM d'un utilisateur DLS homologue	Voir 6.6
Keeper UCMM	Service Lpacket de transfert des DLSDU entre un utilisateur DLS et les objets Keeper d'autres nœuds	Voir 6.7
TUI	Diffusion Lpacket à toutes les stations assistant leur synchronisation à la configuration de liaison en cours	Voir 6.8
Paramètres de liaison	Service Lpacket de publication des paramètres de liaison en attente avant une modification synchronisée des conditions de fonctionnement de liaison	Voir 6.9
tMinus	Service Lpacket de démarrage d'un compte à rebours donnant lieu à une modification des paramètres de liaison et au renvoi du résultat.	Voir 6.9
I'm alive	Diffusion Lpacket par de nouveaux dispositifs réinitialisés afin de faciliter leur entrée dans un système d'exploitation	Voir 6.10
demande et réponse ping	Service Lpacket sollicitant des réponses d'identification de tous les autres dispositifs actifs de la liaison	Voir 6.11
WAMI	Lpacket permettant aux dispositifs temporaires d'identifier un MAC ID de dispositif local lorsqu'ils sont connectés à son port d'accès au réseau	Voir 6.12
débogage	service Lpacket de suivi des transitions d'état de l'objet	Voir 6.13
Objets		
Objet ControlNet	détient et distribue des classes de données de gestion de station et des instances des couches Ph et DL, et collecte des informations de diagnostic à l'intention d'autres entités utilisateur DLS	Voir 7.2

Élément DLL	Description	Emplacement
Objet Keeper	accessible via Keeper UCMM ou la balise fixe UCMM générale, afin de fournir un ensemble de services de support pour les opérations de liaison, les planifications et les connexions détient des données de liaison et des attributs pour tous les dispositifs d'une liaison conserve des copies de toutes les données de l'auteur de la connexion pour les dispositifs à l'origine de la connexion d'un réseau négocie avec d'autres objets Keeper d'une liaison afin de convenir du Keeper maître et des rôles du Keeper de sauvegarde lorsque le Keeper maître distribue des attributs de liaison à tous les nœuds et maintient la synchronisation des Keepers de sauvegarde émet régulièrement TUI Lpacket pour s'assurer que toutes les stations sont synchronisées à la configuration de liaison en cours gère le sémaphore de ressource réseau permettant d'activer la modification régulière des données d'attribut de liaison	Voir 7.3
Objet de planification	offre des services au logiciel de planification de liaison externe en support à la lecture/écriture des informations de connexion et de planification de la part des dispositifs concernés stocke les détails de la planification en cours et le mot de passe de connexion dans Keeper	Voir 7.4
Objet d'interface TCP/IP	fournit le mécanisme de configuration de l'interface TCP/IP d'un dispositif	Voir 7.5
Objet de liaison Ethernet	gère les compteurs spécifiques à la liaison et les informations d'état d'un port Ethernet physique ISO/CEI 8802-3	Voir 7.6
Objet DeviceNet	détient et distribue des classes de données de gestion de station et des instances des couches Ph et DL, et collecte des informations de diagnostic à l'intention d'autres entités utilisateur DLS	Voir 7.7
Objet de configuration de connexion	offre une interface de création, de configuration et de contrôle des connexions d'un dispositif	Voir 7.8

6 Structure DLPDU spécifique, codage et procédures

6.1 Langage de modélisation

6.1.1 Description du diagramme d'état (state machine)

Certains composants de la couche de liaison de données sont spécifiés à l'aide d'un langage de description de diagramme d'état. La déclaration d'action est exprimée en langage C++. Les autres composants ne requièrent pas de description de diagramme d'état et sont spécifiés en C++ uniquement.

Un diagramme d'état est décrit par la syntaxe ci-dessous, où * et {} indiquent respectivement la répétition d'un ou de plusieurs composants précédents et un composant facultatif:

```
state-machine :=
  <header-statements>
  state*
state :=
  "state:" <state-name>
  transition*
transition :=
  "event:" <event-expression>
  { "condition:" <condition-expression> }
  "destination:" <state-name>
  { "action:" <action-statements> }
```

Les noms d'état sont des identifiants C++ normaux, les expressions d'événement et de condition sont des expressions C++, et les déclarations d'en-tête et d'action sont des listes de déclarations C++.

Les déclarations d'en-tête incluent:

- #include et #define;
- les déclarations de variable et d'interface;
- les définitions de sous-routine.

Une expression d'événement indique l'événement spécifié lorsque la valeur de l'expression est TRUE. Un événement doit déclencher une transition. Une expression de condition qualifie une transition. Une transition qualifiée se produit lorsque son événement a lieu si la condition est vraie au moment de l'événement. En général, les expressions d'événement sont FALSE à l'entrée dans un état. Il existe peu de cas d'expressions d'événement vraies (TRUE) à l'entrée dans un état. Dans ces cas, la transition a lieu immédiatement.

Les déclarations d'action sont exécutées lors du déclenchement de la transition.

Tous les déclencheurs d'événement et toutes les conditions de l'état en cours sont évalués en continu et simultanément tant qu'une transition donnée n'est pas activée. Toutes les expressions et déclarations s'exécutent instantanément. Le modèle ci-dessous illustre la priorité des différentes parties d'une transition:

```
if(the event has occurred && the condition is true)
{
  do the actions;
  go to the destination;
}
```

Les mises en œuvre peuvent utiliser un autre code ou une autre syntaxe. Toutefois, les performances de la mise en œuvre doivent être équivalentes à celles spécifiées, sauf pour le déroulement d'exécution interne et les variables locales.

6.1.2 Utilisation du préfixe DLL

Les primitives et paramètres décrits dans la CEI 61158-3-2 utilisent les préfixes suivants:

- DL- pour liaison de données
- DLS- pour service de liaison DL
- DLMS- pour service de gestion DL

Dans ce Type 2: Spécification du protocole de liaison de données, le préfixe générique DLL- est utilisé comme un équivalent au préfixe de description du service approprié.

6.1.3 Types de données

Les types de données utilisés dans les variables de liaison de données, les paramètres, les diagrammes d'état et l'exemple de code sont spécifiés dans la CEI 61158-5-2.

La spécification des types de données correspond à la notation de la CEI 61131-3. Une liste des types de données élémentaires, de leur description correspondante et de certaines de leurs valeurs admises est présentée dans le Tableau 9. Des détails supplémentaires sont spécifiés dans la CEI 61158-6-2.

Tableau 9 – Types de données élémentaires

Mot clé	Description	Plage	
		Minimum	Maximum
BOOL	Booléen		
SINT	Entier court	-128	127
INT	Entier	-32 768	32 767
DINT	Entier double	-2 ³¹	2 ³¹ -1
LINT	Entier long	-2 ⁶³	2 ⁶³ -1
USINT	Entier court non signé	0	255
UINT	Entier non signé	0	65 535
UDINT	Entier double non signé	0	2 ³² -1
ULINT	Entier long non signé	0	2 ⁶⁴ -1
REAL	Virgule flottante		
LREAL	Virgule flottante longue		
ITIME	Durée (court)		
TIME	Durée		
FTIME	Durée (haute résolution)		
LTIME	Durée (long)		
DATE	Date uniquement		
TIME_OF_DAY or TOD	Heure de la journée		
DATE_AND_TIME or DT	Date et heure de la journée		
STRING	Chaîne de caractères (1 octet par caractère)		
STRING2	Chaîne de caractères (2 octets par caractère)		
STRINGN	Chaîne de caractères (N octets par caractère)		
SHORT_STRING	Chaîne de caractères (1 octet par caractère, indicateur de 1 octet)		
SWORD	Chaîne de bits - 8 bits		
WORD	Chaîne de bits - 16 bits		
DWORD	Chaîne de bits - 32 bits		
LWORD	Chaîne de bits - 64 bits		

6.2 Services utilisateur DLS

6.2.1 Généralités

Les services utilisateur DLL fournis à l'utilisateur DLS comportent les interfaces suivantes utilisées pour les besoins spécifiés. Les définitions de service associées et les diagrammes de séquence temporelle sont donnés dans la CEI 61158-3-2.

6.2.2 Service de transfert en mode connecté et sans connexion

6.2.2.1 Codage du service

Les services de demande et de confirmation utilisés pour mettre en file d'attente l'envoi de Lpackets par la DLL doivent se présenter sous la forme ci-dessous.

a) Les transferts en mode connexion utilisent la demande et confirmation de balise générique.

```
DLL_xmit_generic_request(
    IDENTIFIER id,
    USINT Lpacket[],
    UINT Lpacket_size,
    PRIORITY priority,
    USINT gentag[3] );

DLL_xmit_generic_confirm(
    IDENTIFIER id,
    TXSTATUS status );
```

b) Les transferts en mode sans connexion utilisent la demande et confirmation de balise fixe.

```
DLL_xmit_fixed_request(
    IDENTIFIER id,
    USINT Lpacket[],
    UINT Lpacket_size,
    PRIORITY priority,
    USINT service,
    USINT destID );

DLL_xmit_fixed_confirm(
    IDENTIFIER id,
    TXSTATUS status );
```

c) Les transferts en mode connexion utilisent l'indication de balise générique.

```
DLL_rcv_generic_indication(
    USINT Lpacket[],
    UINT Lpacket_size,
    USINT gentag[3] );
```

d) Les transferts en mode sans connexion utilisent l'indication de balise fixe.

```
DLL_rcv_fixed_indication(
    USINT Lpacket[],
    UINT Lpacket_size,
    USINT service,
    USINT sourceID );
```

6.2.2.2 Modes opératoires de demande et de confirmation

Le paramètre `Lpacket` doit contenir la DLSDU sous la forme d'une séquence ordonnée d'octets. Le paramètre `Lpacket_size` doit spécifier le nombre d'octets que contient la matrice `Lpacket`. Le paramètre `QoS_priority` doit spécifier la file d'attente interne dans laquelle la demande doit être placée, et doit prendre la valeur `LOW`, `HIGH` ou `SCHEDULED`. Seuls les `Lpackets` de la file d'attente `SCHEDULED` peuvent être envoyés lors de la partie planifiée de la NUT. La partie non planifiée de la NUT doit permettre de transmettre les `Lpackets` à partir des files d'attente de priorité `QoS HIGH` et `LOW`, et peut être utilisée pour transmettre les `Lpackets` à partir de la file d'attente `SCHEDULED`. Les files d'attente doivent faire l'objet de priorités de sorte que, lors d'une opportunité de transmission non planifiée, la file d'attente `SCHEDULED` est vidée en premier, suivie de la file d'attente de priorité `QoS HIGH`, puis enfin de la file d'attente de priorité `QoS LOW`.

L'IDENTIFIANT, `id`, doit corréler une confirmation à une demande correspondante. Le paramètre `status` doit renvoyer l'état de la tentative de transmission et prendre la valeur `OK`, `TXABORT` ou `FLUSHED`.

Le service `DLL_xmit_fixed_request` met en file d'attente un `Lpacket` avec une balise fixe à deux octets pour la transmission. Le paramètre `service` doit spécifier le premier octet du `Lpacket` à balise fixe. Le paramètre `destID` doit spécifier le deuxième octet, qui doit déterminer le MAC ID du nœud de destination.

Le paramètre restant, `gentag`, du service `DLL_xmit_generic_request` doit spécifier la balise générique sur laquelle est transmis le `Lpacket`.

NOTE 1 L'utilisation d'une ou de plusieurs files d'attente de priorités différentes est un choix de mise en œuvre interne.

NOTE 2 L'assemblage de la structure `Lpacket` associée et le mode opératoire pour l'envoi sur la liaison sont décrits dans leurs articles respectifs (voir 5.3 et 5.4).

6.2.2.3 Modes opératoires d'indication

Ces indications doivent signaler la délivrance d'un `Lpacket` avec une FCS correcte. Seuls les `Lpackets` à balise fixe dont le service a été activé par un appel réussi à

DLL_enable_fixed_request doivent générer un DLL_rcv_fixed_indication. Seuls les Lpackets à balise générique dont la balise générique a été activée par un appel réussi à DLL_enable_generic_request doivent générer un DLL_rcv_generic_indication.

La matrice d'octets Lpacket doit contenir la DLSDU sous la forme d'une séquence ordonnée d'octets. L'entier Lpacket_size doit spécifier le nombre d'octets que contient la matrice Lpacket.

6.2.3 Service de maintenance de la file d'attente

6.2.3.1 Codage du service

Une fois placé en file d'attente, un Lpacket doit rester dans une file d'attente de transmission tant qu'il n'a pas été transmis ou retiré de la file d'attente. Les services de demande et de confirmation utilisés pour retirer un Lpacket de la file d'attente, avant sa transmission, doivent se présenter sous la forme:

```
DLL_flush-requests-by-QoS_request( PRIORITY priority );
DLL_flush-requests-by-QoS_confirm( PRIORITY priority );
DLL_flush_single_request( PRIORITY priority, IDENTIFIER id );
```

6.2.3.2 Modes opératoires de demande et de confirmation

DLL_flush-requests-by-QoS_request doit annuler toutes les transmissions en attente selon la priorité QoS spécifiée. Chaque transmission en attente doit immédiatement prendre fin avec une confirmation de l'état FLUSHED. DLL_flush_single_request doit annuler un Lpacket particulier identifié par l'id utilisé pour transmettre le Lpacket. Il n'est pas utile que ce service dispose d'une confirmation correspondante tant que la confirmation du Lpacket placé dans la file d'attente n'est pas utilisée à cet objet.

6.2.4 Service de filtrage de balise

6.2.4.1 Codage du service

Par défaut, la couche de liaison de données doit uniquement traiter le Lpacket modérateur (balise fixe 0x00), tous les autres Lpackets doivent être écartés. Les autres couches de protocole peuvent autoriser ou interdire la réception de Lpackets particuliers à l'aide des services ci-dessous:

DLL_enable_generic_request(IDENTIFIER id, USINT gentag[3]);	DLL_enable_generic_confirm(IDENTIFIER id, BOOL result);
DLL_disable_generic_request(IDENTIFIER id, USINT gentag[3]);	DLL_disable_generic_confirm(IDENTIFIER id, BOOL result);
DLL_enable_fixed_request(IDENTIFIER id, USINT service);	DLL_enable_fixed_confirm(IDENTIFIER id, BOOL result);
DLL_disable_fixed_request(IDENTIFIER id, USINT service);	DLL_disable_fixed_confirm(IDENTIFIER id, BOOL result);

6.2.4.2 Modes opératoires de demande et de confirmation

L'IDENTIFIANT, id, doit corréler une confirmation à une demande correspondante. Le paramètre, result, doit renvoyer la valeur TRUE si la demande a abouti et la valeur FALSE dans le cas contraire. Le paramètre gentag doit préciser la balise générique à acheminer vers la couche supérieure qui l'a activée. De même, le paramètre service doit préciser la balise fixe à acheminer vers la couche supérieure qui l'a activée.

NOTE Le service DLL_enable_generic_request peut renvoyer un résultat erroné si la mise en œuvre de la couche de liaison de données n'est pas en mesure de filtrer d'autres balises génériques.

6.2.5 Service de synchronisation de liaison

6.2.5.1 Codage du service

L'indication du commencement d'une nouvelle durée de mise à jour du réseau (NUT) doit se présenter sous la forme ci-dessous:

```
DLL_tone_indication( USINT cycle );
```

6.2.5.2 Modes opératoires d'indication

Le paramètre `cycle` doit renvoyer le compteur d'intervalle de la dernière DLPDU modérateur reçue.

NOTE Si une DLPDU modérateur est attendue mais n'arrive pas, le nœud local simule la réception de la DLPDU modérateur en fonction d'un temporisateur interne de l'ACM.

6.2.6 Service de modification de paramètre synchronisé

6.2.6.1 Codage du service

Tous les nœuds d'une liaison doivent conserver deux exemplaires des paramètres de liaison: le paramètre en cours et le paramètre en attente. L'exemplaire en cours des paramètres de liaison doit être utilisé pour le fonctionnement à venir de la couche de liaison de données. L'exemplaire en attente doit être conservé pour permettre une modification synchronisée des paramètres de liaison. Les services de l'utilisateur DLL local qui lit et écrit ces paramètres de liaison doivent se présenter sous la forme:

```
DLL_set_pending_request(      DLL_set_pending_confirm(
    IDENTIFIER id,              IDENTIFIER id,
    DLL_config_data params );   BOOL result );

DLL_set_current_request(      DLL_set_current_confirm(
    IDENTIFIER id,              IDENTIFIER id,
    DLL_config_data params );   BOOL result );

DLL_get_pending_request(      DLL_get_pending_confirm(
    IDENTIFIER id );            IDENTIFIER id,
                                DLL_config_data params );

DLL_get_current_request(      DLL_get_current_confirm(
    IDENTIFIER id );            IDENTIFIER id,
                                DLL_config_data params );
```

The `DLL_config_data` doit contenir les paramètres de liaison et se présenter sous la forme:

```
class DLL_config_data
{
public:
    USINT myaddr;           // the MAC ID of this node
    USINT NUT_length;       // the length of the NUT in 10 µs increments
    USINT smax;             // highest MAC ID allowed to transmit scheduled
    USINT umax;             // highest MAC ID allowed to transmit unscheduled
    USINT slotTime;         // time allowed for line turnaround in 1 µs increments
    USINT blanking;         // time to disable RX after DLPDU in 1,6 µs increments
    USINT gb_start;         // 10 µs intervals from start of guardband to tone
    USINT gb_center;        // 10 µs intervals from start of moderator to tone
    USINT modulus;          // modulus of the interval counter
    USINT gb_prestart;       // transmit cut-off, 10 µs intervals before tone
};                          // may not transmit passed this limit
```

Les services de l'utilisateur DLL local qui demandent et confirment le début d'un compte à rebours `tMinus` doit se présenter sous la forme:

```
DLL-tMinus-start-countdown-request(      DLL-tMinus-start-countdown-confirm(
    IDENTIFIER id,                        IDENTIFIER id,
    USINT start_count );                  BOOL result );
```

NOTE Le service de balise fixe `tMinus` (voir 6.9) est utilisé par le Keeper maître pour demander toutes les stations sur la liaison pour initier le compte à rebours de temps de passage. Il s'agit du mode opératoire de l'utilisateur DLL local pour demander et confirmer la participation locale.

A l'expiration du compte à rebours tMinus, chaque station participante transmet une indication locale à son objet ControlNet pour demander une modification des paramètres de liaison en cours en paramètres de liaison en attente.

```
DLL_tminus_zero_indication( );
```

6.2.6.2 Modes opératoires de demande, de confirmation et d'indication

L'IDENTIFIANT, id, doit corréler une confirmation à une demande correspondante. result doit renvoyer la valeur TRUE si la demande a abouti et la valeur FALSE dans le cas contraire.

Le Lpacket modérateur contient un champ appelé tMinus qui doit être utilisé pour synchroniser la mise à jour des paramètres de liaison en cours. DLL-tMinus_start-countdown-request doit favoriser la participation d'un nœud au compte à rebours tMinus et, s'il s'agit d'un nœud modérateur, doit initialiser le champ tMinus du Lpacket modérateur. Le nœud modérateur doit décrémenter ce champ jusqu'à zéro avant de transmettre chaque modérateur. Lorsque le champ tMinus passe de 1 à 0, la DLL de chaque nœud participant au compte à rebours doit générer un DLL_tminus_zero_indication en local et copier ses paramètres de liaison en attente dans son exemplaire en cours. Si le champ tMinus passe à 0 à partir d'une valeur autre que 1, le compte à rebours doit être annulé et aucun DLL_tminus_zero_indication ne doit être généré.

6.2.7 Service de rapport d'événements

L'indication permettant d'alerter l'entité de gestion de station de différents événements internes à la couche de liaison de données doit prendre la forme ci-dessous:

```
DLL_event_indication(
    DLL_event event,
    USINT      mac_id /*[optional]*/ );
```

Le DLL_event, event, doit être l'un des événements énumérés dans le Tableau 10. Le paramètre mac_id doit être utilisé si (event = DLL_EV_badFrame). Il doit indiquer le MAC ID source du nœud qui a transmis la mauvaise DLPDU.

NOTE La DLPDU ayant été endommagée, le MAC ID source peut ne pas être correct.

Tableau 10 – Événements DLL

Événement DLL	Description
DLL_EV_rxGoodFrame	Une DLPDU correcte a été reçue. Elle doit inclure les DLPDU ne contenant aucune donnée (DLPDU nulles), mais doit exclure les modérateurs
DLL_EV_txGoodFrame	Une DLPDU correcte a été transmise. Elle doit inclure les DLPDU ne contenant aucune donnée (DLPDU nuls), mais doit exclure les modérateurs
DLL_EV_badFrame	Une DLPDU endommagée a été reçue. Le paramètre facultatif, qui est le MAC ID source apparent du nœud en cause, doit également être reporté
DLL_EV_errA	Une DLPDU incorrecte a été reçue sur le canal A du support physique ou une DLPDU correcte a été reçue sur le canal B et la valeur FALSE est toujours attribuée au paramètre <code>ph_frame_indication</code> du canal A
DLL_EV_errB	Une DLPDU incorrecte a été reçue sur le canal B du support physique ou une DLPDU correcte a été reçue sur le canal A et la valeur FALSE est toujours attribuée au paramètre <code>ph_frame_indication</code> du canal B
DLL_EV_txAbort	Une DLPDU transmise s'est terminée par une séquence d'annulation
DLL_EV_NUT_overrun	NUT n'est pas assez importante pour assurer tout le trafic planifié
DLL_EV_dribble	Les Lpackets planifiés n'ont pas été envoyés pendant la durée planifiée
DLL_EV_nonconcurrency	Un événement a été détecté et indique que ce nœud n'est pas synchronisé avec le protocole de contrôle d'accès
DLL_EV_rxAbort	Une DLPDU a été transmise et s'est terminée par une séquence d'annulation
DLL_EV_lonely	N'a pas entendu une DLPDU provenant d'un autre nœud sur la liaison pour 8 NUT
DLL_EV_dupNode	Un autre nœud de la liaison utilise le MAC ID de ce nœud
DLL_EV_noisePulse	<code>ph_lock_indication</code> prend la valeur TRUE, puis la valeur FALSE avant que <code>ph_frame_indication</code> prenne la valeur TRUE, mais <code>ph_lock_indication</code> n'a pas suffisamment pris la valeur TRUE pour indiquer une éventuelle DLPDU endommagée
DLL_EV_collision	La valeur de <code>ph_frame_indication</code> était TRUE lorsque ce nœud était sur le point d'être transmis
DLL_EV_invalidModAddress	Un modérateur a été reçu d'un autre nœud dont le MAC ID n'était pas le plus bas sur la liaison
DLL_EV_rogue	Une DLPDU modérateur a été reçue et ne correspond pas aux informations de configuration de liaison sur ce nœud
DLL_EV_deafness	Impossible d'écouter la DLPDU modérateur, même en présence d'un autre trafic de liaison
DLL_EV_supernode	Un modérateur a été reçu du MAC ID 0

6.2.8 Service de FCS erroné

L'indication permettant d'alerter l'entité de gestion de station qu'une DLPDU reçue contient une FCS non valide doit se présenter sous la forme:

```
DLL_badFCS_indication( CHANNEL channel );
```

Le paramètre `channel` doit indiquer l'entité PhL ayant fourni la DLPDU erronée. Ce paramètre doit être CHA ou CHB, correspondant respectivement à la PhL canal A et à la PhL canal B. Cette indication doit être fournie au maximum une fois par DLPDU erronée par canal.

6.2.9 Service du modérateur en cours

L'indication permettant de signaler à l'entité de gestion de station le nœud qui est actuellement le modérateur doit se présenter sous la forme:

```
DLL_currentMod_indication( USINT macID );
```

Le paramètre `macID` doit indiquer le MAC ID du dernier nœud ayant transmis une DLPDU modérateur valide.

6.2.10 Services de mise sous tension et de mise en ligne

6.2.10.1 Codage du service

La demande et confirmation permettant de placer la couche de liaison de données en ligne doivent se présenter sous la forme:

```
DLL_online_request( BOOL online );          DLL_online_confirm( BOOL online );
```

L'indication qui spécifie que l'initialisation de la couche de liaison de données est terminée doit se présenter sous la forme:

```
DLL_powerup_indication( void );
```

6.2.10.2 Modes opératoires de demande et de confirmation

A la mise sous tension, la DLL doit attendre que la valeur du paramètre `online` soit TRUE. Ensuite, la DLL doit commencer le processus de mise en ligne. Si la valeur du paramètre `online` est FALSE, la DLL doit être mise hors ligne à la fin de la NUT en cours. Hors ligne, la DLL ne doit pas transmettre et doit ignorer les activités de liaison.

6.2.11 Service d'activation du modérateur

6.2.11.1 Codage du service

Les services de demande et de confirmation permettant et interdisant au nœud de jouer le rôle de modérateur doivent se présenter sous la forme:

```
DLL_enable_moderator_request( BOOL enable );
DLL_enable_moderator_confirm( BOOL enable );
```

6.2.11.2 Modes opératoires de demande et de confirmation

Si la valeur du paramètre `enable` est TRUE, la DLL doit activer la transmission des DLPDU modérateurs. Si la valeur du paramètre `enable` est FALSE, la transmission des DLPDU modérateurs doit être désactivée.

NOTE Cette demande est utilisée par les entités de gestion de station pour permettre à un nouveau nœud de joindre une liaison de fonctionnement sans la perturber. Le protocole de basculement du modérateur ne tolère pas le nœud doté du MAC ID le plus bas en supprimant les DLPDU modérateurs pendant une période étendue. Le Moniteur de connexion au réseau (NAM) active de nouveau la transmission du modérateur à chaque fois qu'un autre dispositif est détecté sur la liaison.

6.2.12 Service d'écoute uniquement

6.2.12.1 Codage du service

Les services de demande et de confirmation permettant à un dispositif de recevoir, mais pas de transmettre, doivent se présenter sous la forme:

```
DLL_listen_only_request( BOOL enable );
DLL_listen_only_confirm( BOOL enable );
```

6.2.12.2 Modes opératoires de demande, de confirmation et d'indication

Si la valeur du paramètre `enable` est TRUE, la couche de liaison de données doit participer au protocole d'accès et transmettre des DLPDU. Si la valeur du paramètre `enable` est FALSE, la transmission des DLPDU doit être désactivée. Toutefois, le nœud doit toujours avoir la possibilité de recevoir des DLPDU.

6.3 Lpacket à balise générique

6.3.1 Généralités

Le Lpacket à balise générique permet de transférer des données de connexion. L'ID connexion ou la balise générique est un identifiant unique des ressources et paramètres négociés se trouvant à sa source, aux destinations ou en tout autre point de transit intermédiaire. Seule la balise générique est nécessaire à l'identification des données du message.

6.3.2 Structure du Lpacket à balise générique

Les champs de taille et de contrôle doivent se présenter au format de balise générique. L'adresse doit être la balise générique DLS fournie par le service de demande générique DL local. Le champ de liaison de données doit contenir les données utilisateur DLS fournies par la demande générique DL.

6.3.3 Envoi et réception du Lpacket à balise générique

La station d'envoi doit envoyer le Lpacket au niveau de la priorité QoS spécifiée par la demande générique DL.

L'état de chaque transmission de Lpacket à balise générique est confirmé à l'utilisateur DL par un renvoi de l'ID utilisateur DLS (descripteur ou identifiant interne) et du DLS Txstatus comportant l'une des valeurs ci-dessous.

- a) OK — le message a été envoyé;
- b) TXABORT — le processus d'envoi n'a pas abouti;
- c) FLUSHED — le message a été retiré de la file d'attente avant l'envoi.

NOTE 1 Txstatus est une variable locale. Le codage n'est donc pas spécifié.

NOTE 2 L'état FLUSHED est uniquement utilisé suite à une suppression demandée par le service de maintenance de la file d'attente.

NOTE 3 La valeur de paramètre OK n'indique pas que le message a été reçu.

Toutes les stations destinataires dont le service de filtrage de balise local contient une adresse correspondant à l'adresse Lpacket reçue doivent transmettre chaque unité de données correctement reçue à leur utilisateur DL local, avec la taille et l'adresse de l'unité de données (la balise générique pour la connexion).

6.4 Lpacket modérateur

6.4.1 Généralités

Le Lpacket MAC modérateur est utilisé pour la gestion DLL, la temporisation d'accès et la synchronisation. Il est envoyé en tant que DLPDU modérateur ne contenant qu'un Lpacket: le Lpacket modérateur.

6.4.2 Structure du Lpacket modérateur

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe du modérateur et de 0xFF, l'adresse de diffusion de tous les nœuds MAC IC sur la liaison. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class moderatorLpacket : public fixedLpacket
{
public:
    UINT    NUT_length;    // the length of the NUT in 10 µs increments
    USINT    smax;         // highest MAC ID allowed to transmit scheduled
    USINT    umax;         // highest MAC ID allowed to transmit unscheduled
    USINT    slotTime;     // 1 µs increments allowed for line turnaround
    USINT    blanking;     // 1,6 µs increments to disable RX after DLPDU
    USINT    gb_start;     // 10 µs intervals from start of guardband to tone
    USINT    gb_center;    // 10 µs intervals from start of moderator to tone
    USINT    usr;         // unscheduled start register
    USINT    interval_count; // current NUT number (0 to modulus)
    USINT    modulus;      // modulus of the interval counter
    USINT    tMinus;       // countdown to synchronized link parameter change
    USINT    gb_prestart;  // transmit cut-off, 10 µs intervals before tone
    USINT    reserved;     //
};
```

6.4.3 Envoi et réception du Lpacket modérateur

Le nœud doté du MAC ID le plus bas doit transmettre la DLPDU modérateur contenant le Lpacket modérateur une fois par NUT. Il doit être envoyé dans la bande de garde, partie fixe de chaque NUT réservée au DLPDU modérateur.

Pour prendre en charge les actions de gestion DLL, l'émetteur et les récepteurs de la DLPDU modérateur doivent utiliser le contenu du champ de données de liaison du modérateur, comme suit.

NUT_length doit spécifier la durée de la NUT en cours. NUT_length doit être un entier de 16 bits représentant le nombre d'impulsions de 10 µs et doit être compris entre 50 (500 µs) et 10 000 (100 ms). Le début de la NUT doit être appelé « tonalité ». A la tonalité, un compteur qui décrémente toutes les 10 µs doit être chargé avec NUT_length.

NOTE 1 Le paragraphe 8.2 contraint la valeur de NUT_length.

Pour garantir le calme de la liaison pendant la transmission du modérateur, tous les nœuds doivent cesser de transmettre lorsque le compteur interne atteint la valeur de gb_prestart. Tous les nœuds peuvent alors supposer que la liaison est calme entre gb_start et gb_center. Lorsque le compteur du nœud modérateur atteint la valeur de gb_center, il doit commencer à transmettre une DLPDU contenant uniquement le Lpacket modérateur. Le Lpacket modérateur doit être intégralement transmis au moins 30 µs avant l'expiration du compteur de décrémentation.

NOTE 2 Le paragraphe 8.2 calcule les valeurs de gb_prestart, gb_start et gb_center pour garantir ces contraintes. Les facteurs ayant un impact sur le calcul de ces valeurs incluent la précision d'horloge, les nœuds manquants et le délai de propagation entre les nœuds. Dans tous les cas, $gb\_prestart > gb\_start > gb\_center > 7$.

Le MAC ID le plus élevé assurant la transmission pendant la partie planifiée de la NUT doit être smax. Il doit être compris entre 0 et 254. umax doit déterminer le MAC ID le plus élevé qui assure la transmission pendant la partie non planifiée de la NUT. Il doit être compris entre smax et 254. Le registre de démarrage non planifié, usr, doit sélectionner le nœud qui va transmettre en premier dans la partie non planifiée de la NUT. usr doit incrémenter chaque NUT. Ce compteur doit être un modulo mis à jour (umax + 1).

Un nœud doit écouter la PhL directement à l'issue de la transmission. blanking doit déterminer la durée de suspension d'écoute et doit être dans des impulsions de 1,6 µs.

Le nœud modérateur doit incrémenter un compteur de 8 bits, appelé interval_count, une fois par NUT. Ce compteur doit être un modulo mis à jour (modulus + 1). Étant donné que la bande de garde se trouve à la fin de la NUT, la valeur de interval_count doit correspondre à la NUT qui vient de se terminer.

NOTE 3 Le paragraphe 8.2 et les spécifications de l'objet ControlNet contraignent la valeur de smax, umax, blanking et modulus.

Le champ `tMinus` doit maintenir un compte à rebours tant que tous les nœuds n'ont pas adopté de nouveaux paramètres de liaison (voir 6.9). Le champ `slotTime` doit déterminer le temps d'attente d'un nœud manquant et doit être dans des impulsions de 1,0 μ s. 1,0 μ s supplémentaire doit être ajouté à la valeur de `slotTime`. Le champ `slotTime` doit être compris entre 8 et 254. Les autres valeurs doivent être réservées. Une valeur nulle doit être attribuée à l'octet `reserved`.

6.5 Lpacket de répartition temporelle

6.5.1 Généralités

La DLPDU modérateur offre un marqueur de référence commun synchronisé entre tous les nœuds du réseau. En distribuant et traitant les datations en fonction de la référence au lieu du message de répartition temporelle, les mises en œuvre sont simplifiées sans compromettre l'exactitude. La synchronisation phase-fréquence est inhérente à ce protocole DL à un niveau très élevé d'exactitude. L'exactitude de la synchronisation temporelle utilisant le format de répartition temporelle défini dans le présent paragraphe dépend de la mise en œuvre. Toutefois, elle peut être supérieure à 10 μ s.

6.5.2 Structure du Lpacket de répartition temporelle

6.5.2.1 Format du paquet

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe de répartition temporelle et de l'adresse de diffusion 0xFF. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué:

```
class TimeDist_Lpacket : public fixedLpacket
{
public:
    USINT    revision;    // revision of time distribution format
    USINT    leap;        // leap second offset
    UINT     goodness;    // time relay control field
    DINT     gse;         // global squared error relative to ultimate master
    LINT     dctz;        // distribution channel time zero
    LINT     ts_ref;      // time stamp of previous reference pulse
    LINT     ts_tx;       // time stamp of this message's transmission
};
```

6.5.2.2 Revision

Une valeur nulle doit être attribuée au paramètre `revision`. Ce paramètre doit représenter la révision du format de répartition temporelle.

6.5.2.3 Leap

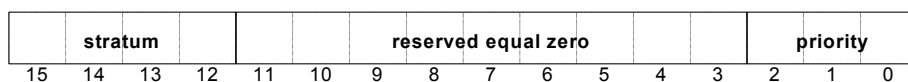
Le paramètre `leap` doit spécifier les secondes intercalaires UTC (Universal Coordinated Time). Ce nombre, lorsqu'il est ajouté au temps système, doit donner le temps UTC réel. Il peut augmenter ou diminuer de 1 deux fois par an, aux solstices, comme indiqué par l'US Naval Observatory, afin de maintenir le synchronisme entre le temps terrestre et le temps astronomique. S'il est nul, le nombre de secondes intercalaires n'est pas connu.

NOTE Il convient de ne pas utiliser le paramètre `leap` dans des situations de contrôle. Cependant, il peut être nécessaire dans certains relais temporels de canaux de distribution reposant sur le temps UTC plutôt que GPS (Global Positioning Satellite).

6.5.2.4 Goodness

6.5.2.4.1 Structure de paramètre

Le paramètre `goodness` doit être partitionné comme indiqué dans la Figure 15.



Légende

Anglais	Français
stratum	strate
reserved equal zero	réservés, valeur nulle
priority	priorité

Figure 15 – Paramètre goodness de TimeDist_Lpacket

6.5.2.4.2 Strate

Le champ le plus significatif, appelé strate, doit indiquer le nombre de relais temporels entre ce message et une source de temps absolu. Une valeur nulle doit signifier une référence exacte, et la valeur doit être incrémentée pour tous les relais temporels intermédiaires. Si une valeur nulle est attribuée au champ de priorité (verrouillage non obtenu) ou que le nombre de relais temporels intermédiaires est supérieur à 15, la valeur 15 doit être attribuée au paramètre d'exactitude. Les bits 3 à 11 doivent être réservés et nuls.

NOTE Un relais temporel est un routeur liaison à liaison qui distribue les messages de synchronisation temporelle de sa liaison en fonction des messages de synchronisation temporelle de ses autres liaisons.

6.5.2.4.3 Qualité de la source

Le champ de priorité doit indiquer la qualité du message temporel (voir le Tableau 11).

Tableau 11 – Priorité de répartition temporelle

Valeur	Signification
7	Temps système absolu. Fait office de maître
6	Temps système absolu. Fait office de subordonné
5	Temps système défini par l'homme. Fait office de maître
4	Temps système défini par l'homme. Fait office de subordonné
3	Verrou appliqué au nœud du canal de distribution autre que celui-là
2	Verrou appliqué au nœud de ce canal. Temps système inconnu
1	(non valide)
0	Non synchronisé avec un autre nœud

6.5.2.5 gse

Le paramètre gse doit indiquer la stabilité rms cumulative au carré. Ce paramètre doit approximer la stabilité du cas le moins favorable du nœud par rapport au reste du système. Les unités de ce paramètre doivent être $(0,1 \mu s)^2$. Si la stabilité rms est inconnue ou pas encore déterminée, elle doit être de 0xFFFFFFFF.

6.5.2.6 Paramètres temporels LINT

Dans chaque paramètre temporel LINT, le bit de poids fort doit être nul (uniquement 63 bits doivent être utilisés). Les unités de ces paramètres doivent être $0,1 \mu s$.

6.5.2.7 **dctz**

Le paramètre **dctz** doit indiquer le décalage de temps système du temps zéro arbitraire du canal de distribution, établi lors de la synchronisation des réseaux.

6.5.2.8 **ts_ref**

Le paramètre **ts_ref** doit indiquer la datation de la dernière tonalité suivant une DLPDU modérateur dont l'intervalle **interval_count** est nul. La valeur nulle doit indiquer que cette valeur est inconnue. Le temps zéro du système doit être défini comme celui étant utilisé pour les systèmes GPS: 6 janvier 1980 12:00 minuit GMT.

6.5.2.9 **ts_tx**

Le paramètre **ts_tx** doit indiquer la datation au niveau de la transmission de ce message. La valeur nulle doit indiquer que cette valeur est inconnue. Le temps zéro du système doit être défini comme celui étant utilisé pour les systèmes GPS: 6 janvier 1980 12:00 minuit GMT. Une valeur nulle doit être attribuée à ce paramètre.

NOTE Ce protocole n'utilise pas le temps GPS, mais uniquement le temps zéro GPS. Par conséquent, 1999 GPS n'a aucun intérêt pour le temps système.

6.5.3 **Envoi et réception du Lpacket de répartition temporelle**

Le Keeper maître en cours est chargé d'envoyer le Lpacket de répartition temporelle à des intervalles et des valeurs QoS configurés par l'utilisateur DL en fonction des exigences de son application.

Tous les nœuds sont chargés de recevoir le Lpacket de répartition temporelle.

6.6 **Lpacket UCMM**

6.6.1 **Généralités**

L'UCMM (Unconnected Message Manager) est une entité utilisateur DLS offrant un service de message de demande/réponse sans connexion pour au moins un message en attente. Elle utilise le service de transmission DLL et le Lpacket UCMM pour ses messages.

6.6.2 **Structure du Lpacket UCMM**

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe UCMM et du MAC ID de destination fourni par le service de transmission DLL. Le champ de liaison de données doit contenir les données utilisateur DLS fournies par le service de transmission DLL.

6.6.3 **Envoi et réception du Lpacket UCMM**

Le Lpacket UCMM doit être envoyé à la priorité QoS spécifiée par le service de transfert de données DLL.

L'état de chaque transmission UCMM est confirmé à l'utilisateur DL par un renvoi de l'ID utilisateur DLS (descripteur ou identifiant interne) et du DLS Txstatus comportant l'une des valeurs ci-dessous.

- a) OK — le message a été envoyé;
- b) TXABORT — le processus d'envoi n'a pas abouti;
- c) FLUSHED — le message a été retiré de la file d'attente avant l'envoi.

NOTE 1 Txstatus est une variable locale. Le codage n'est donc pas spécifié.

NOTE 2 L'état FLUSHED est uniquement utilisé suite à une suppression demandée par le service de maintenance de la file d'attente.

NOTE 3 La valeur de paramètre OK n'indique pas que le message a été reçu.

La station destinataire dont le MAC ID correspond à celui de l'adresse doit transmettre chaque unité de données correctement reçue à son utilisateur DL local, avec la taille de l'unité de données, le type de service Lpacket et son ID source obtenu auprès de la DLPDU qui a acheminé le Lpacket.

6.7 Lpacket Keeper UCMM

6.7.1 Généralités

Les dispositifs qui mettent en œuvre l'objet Keeper doivent également prendre en charge l'envoi et la réception des Lpackets Keeper UCMM. Seul l'objet Keeper maître doit envoyer le Lpacket Keeper UCMM.

NOTE L'utilisation d'une adresse spécifique pour les objets Keeper permet de réduire l'activité de décodage requise par les dispositifs qui ne contiennent pas d'objet Keeper.

6.7.2 Structure du Lpacket Keeper UCMM

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe Keeper UCMM et de l'adresse de diffusion 0xFF. Le champ de liaison de données doit contenir les données utilisateur fournies par l'objet Keeper.

6.7.3 Envoi et réception du Lpacket Keeper UCMM

Le Lpacket Keeper UCMM doit uniquement être envoyé par une station dont l'objet Keeper est le maître. Il doit être demandé à la priorité QoS spécifiée par l'objet Keeper appelant.

L'état de chaque transmission Keeper UCMM est confirmé à l'utilisateur DL par un renvoi de l'ID utilisateur DLS (descripteur ou identifiant interne) et du DLS Txstatus comportant l'une des valeurs ci-dessous.

- a) OK — le message a été envoyé;
- b) TXABORT — le processus d'envoi n'a pas abouti;
- c) FLUSHED — le message a été retiré de la file d'attente avant l'envoi.

NOTE 1 Txstatus est une variable locale. Le codage n'est donc pas spécifié.

NOTE 2 L'état FLUSHED est uniquement utilisé suite à une suppression demandée par le service de maintenance de la file d'attente.

NOTE 3 La valeur de paramètre OK n'indique pas que le message a été reçu.

La station destinataire dont le MAC ID correspond à celui de l'adresse doit transmettre chaque unité de données correctement reçue à son utilisateur DL local, avec la taille de l'unité de données, le type de service Lpacket et son ID source obtenu auprès de la DLPDU qui a acheminé le Lpacket.

NOTE 4 Étant donné qu'il s'agit d'un message sans connexion, le MAC ID source est requis pour activer une réponse appropriée.

6.8 Lpacket TUI

6.8.1 Généralités

Un Identifiant unique de table (TUI) est utilisé par tous les objets ControlNet pour faire référence à un ensemble cohérent d'attributs de configuration de liaison. Deux ensembles distincts d'attributs sont traités, chacun doté de leur propre TUI: l'un pour le fonctionnement en cours et l'autre pour le fonctionnement en attente. Le passage des données TUI de la

configuration de liaison en cours vers celles de la configuration de liaison en attente doit avoir lieu à l'issue d'une modification de paramètre synchronisé.

Le Lpacket TUI est publié comme un Lpacket planifié par le Keeper maître pour activer tous les dispositifs connexes afin de s'assurer qu'ils comportent des données de configuration et de planification en cours.

6.8.2 Structure du Lpacket TUI

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe TUI et de l'adresse de diffusion 0xFF. Le champ de données de liaison doit contenir les éléments suivants du Tableau 12.

Tableau 12 – Format du Lpacket TUI

Nom	Type de données	Description du paramètre	Sémantique des valeurs
Taille	USINT	Taille du Lpacket, en mots	0x0D
Contrôle	USINT	Octet de contrôle du Lpacket de couche de liaison	0x01 (Lpacket à balise fixe)
Fixed_Tag	USINT	Valeur de balise fixe	0x84 (Lpacket TUI)
Destination_Mac_Id	USINT	Destinataire du Lpacket	0xFF (Diffusion)
Unique_Id	UDINT	CRC des attributs en cours de Keeper	Octet de poids faible en premier
Status_Flag	UINT	Valeurs de balise TUI	Voir l'attribut TUI de l'objet Keeper pour plus de détails
Keeper_Mac_Id	USINT	MAC ID du Keeper diffusant le TUI	Voir l'attribut TUI de l'objet Keeper pour plus de détails
Réservé	USINT	Réservé pour l'alignement des données	
Net_Resource_Vendor_Id	UINT	ID fournisseur de l'objet détenant le Net Resource pour une utilisation exclusive	
Net_Resource_Serial_Number	UDINT	Numéro de série de l'objet détenant le Net Resource pour une utilisation exclusive	
Net_Resource_Class	UDINT	Numéro de classe de l'objet détenant le Net Resource pour une utilisation exclusive	
Net_Resource_Instance	UDINT	Numéro d'instance de l'objet détenant le Net Resource pour une utilisation exclusive	

Lors de la génération du Lpacket TUI transmis, tous les champs réservés doivent utiliser les valeurs spécifiées dans les champs réservés correspondants de l'attribut d'objet Keeper 0x0101. Lors du processus de configuration, l'outil de programmation doit affecter une valeur nulle à tous les champs réservés de l'attribut 0x0101.

6.8.3 Envoi et réception du Lpacket TUI

La priorité QoS demandée doit être planifiée.

Si l'objet Keeper est à l'état de fonctionnement MASTER_VERIFY, FAULTED_MASTER_VERIFY, MASTER, FAULTED_MASTER, NET_CHANGE_MASTER ou NET_CHANGE_FAULTED_MASTER, il doit tenter de transmettre un Lpacket TUI en tant que données planifiées une fois toutes les 4 NUT (NUT 0, 4, 8, 12, 16 ...). Tous les dispositifs Keeper doivent réserver 26 octets de

durée de transmission de liaison planifiée une fois toutes les 4 NUT pour transmettre ce Lpacket (en cas d'envoi non planifié, il doit s'agir des informations non planifiées de priorité QoS la plus élevée).

La réception des Lpackets TUI doit être activée à la mise sous tension par l'objet ControlNet dans chaque dispositif. Les Lpackets TUI réservés doivent être transmis à l'objet ControlNet local.

6.9 Paramètres de liaison Lpacket et tMinus Lpacket

6.9.1 Généralités

Pour synchroniser la modification des paramètres de liaison locaux, l'entité de gestion de station doit activer la réception des deux Lpackets à balise fixe: 0x15 (tMinus) et 0x81 (paramètres de liaison) via le service DLL_enable_fixed_request de la DLL.

Tous les nœuds d'une liaison doivent conserver deux exemplaires des paramètres de liaison: le paramètre en cours et le paramètre en attente. L'exemplaire en cours des paramètres de liaison doit être utilisé pour le fonctionnement à venir de la couche de liaison de données. L'exemplaire en attente doit être conservé pour permettre une modification synchronisée des paramètres de liaison. La synchronisation de modification est prise en charge par le service tMinus.

6.9.2 Structure des paramètres de liaison et des Lpackets tMinus

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe pour le service et de l'adresse de diffusion 0xFF. Le champ de données de liaison du Lpacket approprié doit contenir les éléments suivants dans l'ordre indiqué.

a) Contenu du champ de données Lpacket des paramètres de liaison:

```
class LinkParm_Lpacket : public fixedLpacket
{
public:
    UINT NUT_length; // the length of the NUT in 10 µs increments;
    USINT smax; // highest MAC ID allowed to transmit scheduled
    USINT umax; // highest MAC ID allowed to transmit unscheduled
    USINT slotTime; // time allowed for line turnaround in 1 µs increments
    USINT blanking; // time to disable RX after DLPDU in 1,6 µs increments
    USINT gb_start; // 10 µs intervals from start of guardband to tone
    USINT gb_center; // 10 µs intervals from start of moderator to tone
    UINT zero; // reserved
    USINT modulus; // modulus of the interval counter
    USINT gp_prestart; // transmit cut-off, 10 µs intervals before tone
    UDINT unique_id; // 32 bit CRC calculated from Keeper configuration table
    UINT status_flag; // 16 bit status table set by the Keeper
    UINT reserved[8]; // all zeros
};
```

NOTE Le format des attributs d'instance 0x80 et 0x81 de l'objet ControlNet est exactement identique aux données de liaison du Lpacket 0x81 à balise fixe.

b) Contenu du champ de données Lpacket tMinus:

```
class tMinus_Lpacket : public fixedLpacket
{
public:
    USINT start_count;
    USINT reserved[3];
};
```

où le start_count est le nombre de NUT à compter avant la transition et ne doit pas être inférieur à 8, le champ réservé doit contenir des zéros.

6.9.3 Envoi et réception des Lpackets tMinus et des paramètres de liaison

Les paramètres de liaison et les Lpackets tMinus sont émis par le Keeper maître lorsqu'une modification des paramètres de liaison est requise. Ils doivent être demandés à la priorité QoS spécifiée par l'objet Keeper appelant.

A la réception de LinkParm_Lpacket, chaque nœud doit charger l'exemplaire en attente de ses paramètres de liaison à l'aide de son service DLL_set_pending_request local. Les paramètres en attente doivent être chargés en 3 opportunités de transmission non planifiées ou en 1 s, selon la valeur la plus élevée. L'attribut de paramètre de liaison en attente de l'objet ControlNet doit également être mis à jour avec l'exemplaire en attente des paramètres de liaison.

A la réception d'un tMinus_Lpacket, le nœud doit lancer une modification de paramètre synchronisé à l'aide du service DLL-tMinus-start-countdown-request transmettant le paramètre start_count. L'objet ControlNet doit copier son attribut de paramètre de liaison en attente dans son attribut de paramètre de liaison en cours lorsque DLL_tminus_zero_indication signale la fin de la modification de paramètre synchronisé.

NOTE La DLL copie ses propres paramètres de liaison en attente dans l'exemplaire en cours à l'issue du compte à rebours tMinus, aucun DLL_tminus_zero_indication n'étant signalé, sauf si la fin est précédée d'un DLL-tMinus-start-countdown-request.

Le Lpacket modérateur contient un champ appelé tMinus qui doit être utilisé pour synchroniser la mise à jour des paramètres de liaison en cours. DLL-tMinus-start-countdown-request doit favoriser la participation d'un nœud au compte à rebours tMinus et, s'il s'agit d'un nœud modérateur, doit initialiser le champ tMinus du Lpacket modérateur. Le nœud modérateur doit décrémenter ce champ jusqu'à zéro avant de transmettre chaque modérateur. Lorsque le champ tMinus passe de 1 à 0, la DLL de chaque nœud participant au compte à rebours doit générer un DLL_tminus_zero_indication en local dans son objet ControlNet, lequel doit copier ses paramètres de liaison en attente dans son exemplaire en cours. Si le champ tMinus passe à 0 à partir d'une valeur autre que 1, le compte à rebours doit être annulé et aucun DLL_tminus_zero_indication ne doit être généré.

6.10 Lpacket l'm-alive

6.10.1 Généralités

Ce service permet à tous les nœuds d'une liaison de reconnaître qu'un autre nœud vient d'être réinitialisé ou activé.

6.10.2 Structure du Lpacket l'm-alive

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe l'm-alive et de l'adresse de diffusion 0xFF. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class Im_Alive_Lpacket : public fixedLpacket
{
public:
    USINT    Lpacket_variant;
    USINT    sourceID;
    USINT    reserved[6];
};
```

Les champs Lpacket_variant et reserved doivent être nuls. Le MAC ID du nœud qui transmet le Lpacket doit être attribué au champ sourceID.

6.10.3 Envoi et réception de l'm Alive

La priorité QoS doit être spécifiée par l'utilisateur DL appelant.

Lorsqu'un nœud est d'abord mis en ligne avant de commencer des opérations complètes, le moniteur de connexion au réseau (NAM) du nouveau nœud doit diffuser au moins 3 Im_Alive_Lpacket, soumis aux règles de l'm Alive de traitement de l'état.

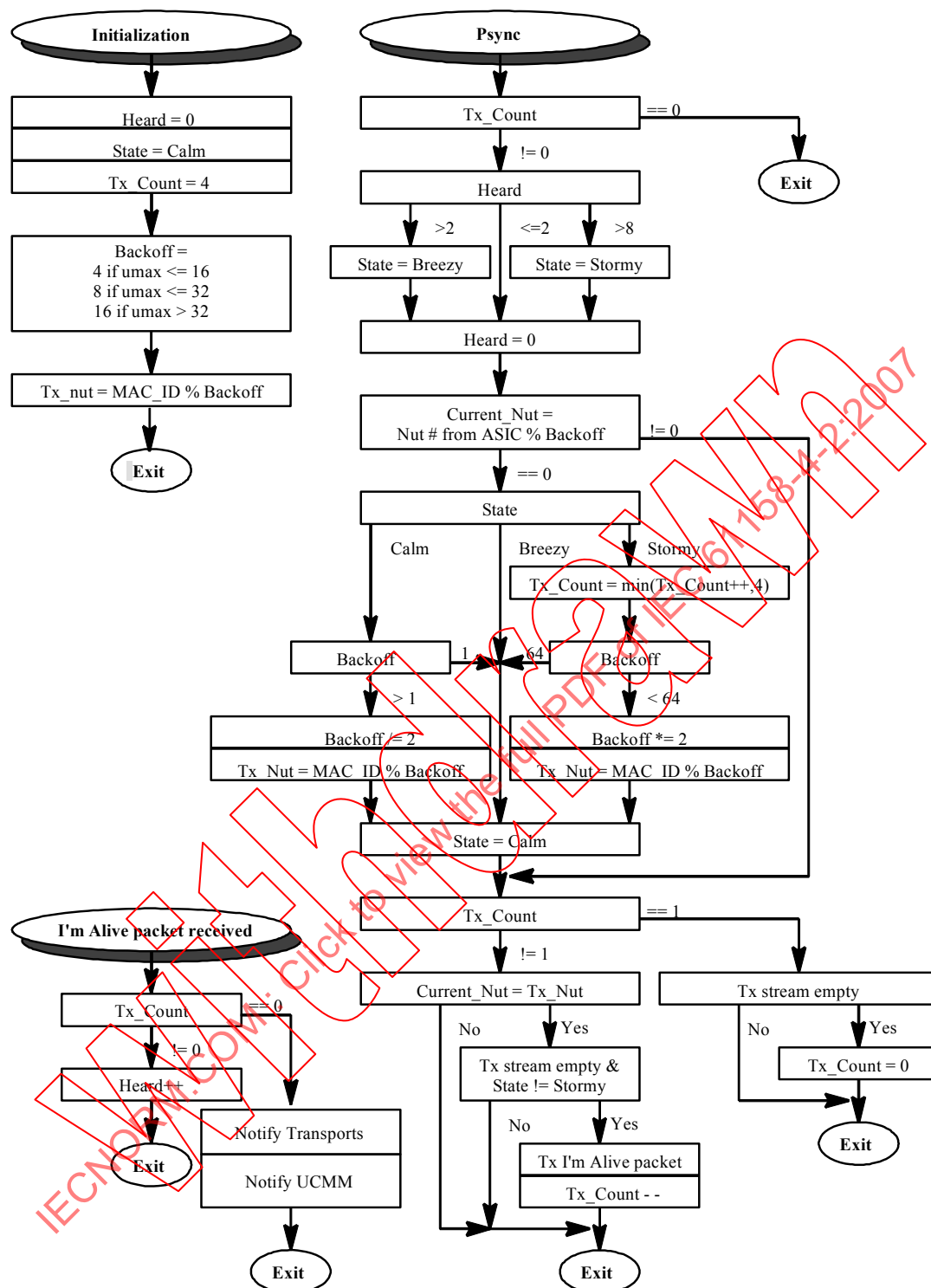
Les stations destinataires doivent informer leurs services de gestion.

NOTE Parmi les clients de ces diffusions se trouvent les gestionnaires de connexion de la couche supérieure de chaque nœud de la liaison. Les entités annulent tous les enregistrements de transaction associés à un MAC ID qui vient d'être activé, ainsi que toutes les connexions transmises par son intermédiaire. Étant donné que les ID connexion (balises génériques) sont attribués de manière dynamique, cela garantit que l'ID connexion émis par l'ancien dispositif au niveau du MAC ID n'est pas réutilisé improprement.

6.10.4 Traitement d'état l'm alive

Un nœud ne doit transmettre qu'un Im_Alive_Lpacket par NUT. Pour limiter les exigences de traitement dans chaque nœud, un nœud qui diffuse ses trois Im_Alive_Lpacket ne doit pas considérer valider la transmission de tous les Im_Alive_Lpacket précédés dans la NUT de plus de sept autres Lpackets de ce type. Ces Lpackets doivent être transmis dans la partie non planifiée de la NUT (priorité QoS LOW ou HIGH).

NOTE Étant donné qu'il est impossible de contrôler le moment où les nœuds sont activés ou réinitialisés les uns par rapport aux autres, il existe une possibilité de saturation des transmissions de Lpacket l'm alive si un grand nombre de nœuds entrent simultanément à l'état l'm alive. Pour limiter les exigences de traitement, un brouillage de transmission et un algorithme de réduction de puissance sont recommandés pour surveiller et contrôler l'intensité d'une saturation des transmissions. Ce type d'algorithme diffuse les Lpackets l'm alive sur des périodes de plus en plus longues, pourvu qu'un grand nombre de nœuds soient activés simultanément. Si peu de nœuds sont activés simultanément, il convient que l'algorithme leur permette de quitter rapidement l'état l'm alive étant donné qu'il ne peut pas y avoir de saturation des transmissions. Un exemple de ce type d'algorithme est présenté dans la Figure 16.



Légende

Anglais	Français
initialization	initialisation
heard	entendu
state	état
exit	sortie
backoff	réduction de puissance
.. if ...	... si ...
I'm Alive packet received	paquet I'm Alive reçu

Anglais	Français
notify	notifier

Figure 16 – Exemple d'algorithme de traitement l'm alive

6.11 Lpackets ping

6.11.1 Généralités

Le service ping permet à l'entité de gestion de station de lancer une demande ping et de recevoir une réponse ping d'une autre station active ou d'une station adressée associée au fournisseur DLS.

Toutes les stations contiennent le service ping et, lors de la mise sous tension, le service enable-fixed-request est utilisé par l'utilisateur DLS afin d'activer la réception et la réponse aux demandes ping dans le fournisseur DLS local.

6.11.2 Structure des Lpackets ping

Les champs de taille et de contrôle doivent se présenter au format de balise fixe.

- a) Lors de l'envoi d'une demande ping, l'adresse doit être composée de la balise fixe de demande ping et d'une adresse spécifiée par l'utilisateur DL. L'adresse spécifiée peut être un MAC ID unique ou l'adresse de diffusion 0xFF.

Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class Ping_request : public fixedLpacket
{
public:
    USINT    client_ID;
    USINT    reserved[3];
};
```

où client_ID est le MAC ID de l'auteur de la demande, et le champ réservé doit contenir des zéros.

- b) Lors de l'envoi d'une réponse ping, l'adresse doit être composée de la balise fixe de réponse ping et de l'adresse client_ID provenant de l'indication de demande ping correspondante (utilisée ici comme destination de la réponse).

Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué;

```
class Ping_reply : public fixedLpacket
{
public:
    USINT    server_ID;
    USINT    vendor_specific[4];
    USINT    reserved[3];
};
```

où le champ server_ID doit contenir le MAC ID du nœud répondant à la demande, où le champ vendor_specific peut contenir des données et où le champ réservé doit contenir des zéros.

Le service `DLL_xmit_fixed_request(id, Lpacket, sizeof(Ping_reply), HIGH, 0x29, client_ID)` doit être utilisé pour envoyer la réponse au MAC ID spécifié dans le champ client_ID du Lpacket Ping_request reçu.

NOTE En général, le champ vendor_specific est utilisé pour les informations de débogage ou de diagnostic.

6.11.3 Envoi et réception des Lpackets ping

La priorité QoS d'une demande ping doit être spécifiée par l'utilisateur DL appelant. La priorité QoS d'une réponse ping doit être High (Élevée).

Les Lpackets ping-request peuvent être envoyés par une entité de gestion locale dans une station.

Lors de la mise sous tension, le service interne `DLL_enable_fixed_request(id, 0x09)` de la couche de liaison de données doit être appelé pour activer la réception des demandes ping à balise fixe.

Lors de la réception d'une demande ping, chaque station destinataire compose un Lpacket à balise fixe de réponse ping et l'envoie à la station demandeuse.

Lors de la réception d'une réponse ping, le nœud demandeur transmet l'indication à l'entité de gestion locale.

6.12 Lpacket WAMI

6.12.1 Généralités

Un serveur WAMI (Where Am I?) doit être présent dans tous les nœuds permanents dotés d'un port d'accès au réseau (NAP). Le serveur ne doit pas être présent sur les nœuds transitoires.

NOTE Le serveur WAMI (where am I) permet à un nœud transitoire (un nœud qui se connecte au réseau par l'intermédiaire du port d'accès au réseau) de déterminer le MAC ID du nœud auquel il est associé. Cela est très pratique pour les logiciels d'un dispositif transitoire qui souhaitent établir une communication avec le nœud permanent local.

6.12.2 Structure du Lpacket WAMI

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe WAMI et de 0xFF, l'adresse de diffusion de tous les nœuds MAC IC sur la liaison. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class WAMI_Lpacket : public fixedlpacket
{
public:
    USINT    requestID;
    USINT    responseID;
    USINT    reserved[6];
};
```

Dans une demande WAMI, le `requestID` doit contenir le MAC ID du nœud transitoire qui envoie le Lpacket. Les autres champs doivent contenir des zéros. Pour convertir une demande en réponse, le serveur doit copier son MAC ID dans le champ `responseID`, puis répondre. La priorité QoS est Low (Basse).

NOTE 1 Lorsque le service `DLL_enable_fixed_request(id, 0x86)` de la DLL active une réception de balise fixe WAMI, le service `DLL_recv_fixed_indication` reçoit les balises fixes.

NOTE 2 Le Moniteur de connexion au réseau (voir 8.1) décrit les modes opératoires visant à s'assurer que le MAC ID transitoire ne duplique pas un MAC ID de liaison existant.

6.12.3 Envoi et réception du Lpacket WAMI

La priorité QoS doit être Low (Basse).

Le serveur WAMI est uniquement mis en œuvre dans les dispositifs dotés d'un NAP, ces dispositifs doivent répéter tous les messages entre la liaison principale et le NAP, et inversement. De plus, ces dispositifs doivent comporter des moyens de détection des messages reçus par l'intermédiaire de leur NAP, et doivent uniquement répondre aux Lpackets respectant chacun des critères ci-dessous:

- Balise fixe WAMI 0x86;
- diffusion de destination (0xFF);
- reçu par l'intermédiaire du NAP local du dispositif.

Les Lpackets répondant à ces critères doivent être appelés Lpackets WAMI de demande. Pour ce type de Lpackets, le serveur doit générer une réponse WAMI adressée au MAC ID qui envoie la demande WAMI. Le serveur WAMI doit ignorer tous les autres Lpackets 0x86 à balise fixe WAMI.

6.13 Lpacket Debug

Ce service peut être utilisé pour transmettre l'état d'un objet en générant une trace des transitions d'état d'objet sur le réseau.

Les champs de taille et de contrôle doivent se présenter au format de balise fixe. L'adresse doit être composée de la balise fixe Debug et du MAC ID de destination fourni par le service de transmission DLL. Le champ de données de liaison doit contenir les éléments suivants dans l'ordre indiqué.

```
class Debug_Lpacket : public fixedLpacket
{
public:
    UINT    object_class;
    UINT    data1;
    UINT    data2;
};
```

La valeur de `object_class` doit être le code de classe (voir CEI 61158-6-2) de l'objet qui transmet son état. La signification des champs `data1` et `data2` doit être définie dans la définition d'objet. Les dispositifs ne doivent pas générer plusieurs Lpackets Debug à balise fixe toutes les 10 s en moyenne, de manière à ne pas gêner les autres transmissions non planifiées.

6.14 Lpacket IP

Ce service (balise fixe 0x85) permet d'envoyer des trames IP brutes. La taille maximale de trame IP qui peut être envoyée dans un Lpacket est de 506 octets. Les diffusions sont envoyées avec la destination 0xFF. Les trames IP entre deux nœuds doivent comporter le bon MAC ID de destination à balise fixe. Un protocole de résolution d'adresse doit être utilisé afin de déterminer le MAC ID associé à une adresse IP particulière.

6.15 Lpacket Ethernet

Ce service (balise fixe 0x89) permet d'envoyer des trames Ethernet autres que des trames IP. Il s'agit, sans s'y limiter, de trames Ethernet ARP (Address Resolution Protocol). Cela permet de mettre simplement en œuvre un protocole IP et un protocole de résolution d'adresse de style Ethernet sur la couche physique de Type 2, une pile TCP/IP/Ethernet existante pouvant être utilisée avec de petites modifications de paquet.

L'octet le plus bas de l'adresse physique Ethernet est utilisé pour représenter le MAC ID de Type 2 (les adresses physiques Ethernet sont envoyées avec l'octet le plus élevé en premier). Une diffusion de Type 2 (MAC ID 0xFF) doit être étendue à une adresse de diffusion Ethernet (FF FF FF FF FF FF).

Lors du démarrage, une pile TCP/IP contenant Ethernet ARP doit connaître l'adresse physique du nœud local. Il est reporté sous la forme 00 00 00 00 00 <MACID>.

L'octet bas de l'adresse de destination d'une trame Ethernet fait office de destination à balise fixe de Type 2. L'octet bas de l'adresse source d'une trame Ethernet est identique au MAC ID du nœud local (signalé à la pile TCP/IP au démarrage) et est envoyé comme MAC ID source de la DLPDU. Le « type de trame » Ethernet (08 06 pour ARP) compose les deux premiers octets de la partie données du Lpacket, et jusqu'à 504 octets sont disponibles pour la partie données de la trame Ethernet (demande/réponse ARP utilise 28 octets).

Étant donné que le « type de trame » Ethernet est inclus dans le Lpacket, d'autres types de trames Ethernet (RARP, par exemple) peuvent également être envoyés à l'aide de ce service à balise fixe Ethernet.

7 Objets de gestion de la station

7.1 Généralités

Cette spécification de protocole est assez souple. Elle peut assurer un transfert E-S déterministe et synchronisé à intervalles cycliques (toutes les 1 ms) et des séparations de nœud jusqu'à 25 km. Ces performances peuvent être ajustées en ligne en configurant les paramètres de liaison. Ces paramètres, qui régissent l'accès à la liaison, peuvent être réglés pour correspondre à des applications différentes.

La gestion de station permet de modifier ces paramètres en ligne, le réseau fonctionnant, elle permet également à la liaison de continuer à fonctionner en cas d'ajout ou de retrait de nœuds.

Les fonctions de gestion de station permettent:

- d'accéder aux variables et événements de chacune des couches;
- d'utiliser une interface utilisateur commune;
- de coordonner la modification des paramètres de liaison;
- d'ajouter des nœuds à la liaison sans interruption;
- d'ajuster les paramètres de liaison;
- d'assurer la synchronisation d'horloge entre les nœuds.

L'accès aux variables et événements dans chacune des couches est possible en définissant les interfaces d'objet de ces paramètres.

L'objet ControlNet offre une interface cohérente aux couches de liaison physiques et de données.

L'objet Keeper détient tous les attributs de liaison de tous les dispositifs d'une liaison et doit être chargé de les distribuer aux dispositifs de manière ordonnée. Il détient également (pour le logiciel de planification de liaison) un exemplaire des données de l'auteur de la connexion de tous les dispositifs à l'origine de la connexion utilisant un réseau. Si plusieurs objets Keeper sont présents sur une liaison, ils procèdent à des négociations afin de déterminer celui qui va devenir maître et ceux qui vont assurer les fonctions de sauvegarde. Le Keeper maître est celui qui distribue activement les paramètres de liaison et les attributs aux nœuds du réseau. Un objet Keeper de sauvegarde est celui qui surveille l'activité réseau associée au Keeper et peut assurer le rôle d'objet Keeper maître en cas de défaillance de celui-là.

L'objet Keeper contient également un support pour l'obtention, la gestion et la délivrance de la ressource réseau, sémaphore réseau utilisé pour éliminer les conflits lors de la modification des données d'attribut détenues par le/les objet(s) Keeper sur une liaison.

L'objet de planification doit exister dans les dispositifs à l'origine de la connexion (CO). L'objet de planification doit être utilisé par un logiciel de planification de liaison (LSS) pour

- lire les informations de connexion planifiée;
- écrire les informations de planification;
- offrir une signature (clé logicielle) à l'auteur de la connexion, qui doit être utilisée pour authentifier l'accès du dispositif CO à une liaison particulière.

L'objet d'interface TCP/IP fournit le mécanisme de configuration de l'interface TCP/IP d'un dispositif.

L'objet de liaison Ethernet gère les compteurs spécifiques à la liaison et les informations d'état d'un port Ethernet physique ISO/CEI 8802-3.

L'objet DeviceNet offre une interface cohérente aux couches de liaison physiques et de données.

L'objet de configuration de connexion offre une interface de création, de configuration et de contrôle des connexions d'un dispositif.

7.2 Objet ControlNet

7.2.1 Présentation

L'objet ControlNet doit fournir une interface cohérente de gestion de station aux couches de liaison physiques et de données. Cet objet doit mettre les informations de diagnostic de ces couches à la disposition des applications client. Chaque nœud doit prendre en charge un objet ControlNet par liaison.

NOTE Un dispositif peut contenir plusieurs nœuds.

7.2.2 Attributs de classe

L'objet ControlNet doit prendre en charge les attributs de classe spécifiés dans le Tableau 13.

Tableau 13 – Attributs de classe de l'objet ControlNet

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x01	obligatoire	Get	Revision	UINT	Révision de cet objet	Première révision, valeur = 1
0x02	obligatoire	Get	Max. instance	UDINT	Nombre maximal d'instances	Valeur déterminée par les caractéristiques du nœud

7.2.3 Attributs d'instance

7.2.3.1 Généralités

L'objet ControlNet doit prendre en charge les attributs d'instance spécifiés dans le Tableau 14.

Tableau 14 – Attributs d'instance de l'objet ControlNet

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x80	Facultatif	Get	Pending_Link_Config	STRUCT de 34 octets	paramètres de configuration de liaison en attente	voir 6.9.2
			Link_Config	STRUCT de 12 octets	paramètres de liaison en attente	
			NUT_length	UINT	DLL NUT_length	dans des impulsions de 10 µs
			smax	USINT	DLL smax	0 à 99
			umax	USINT	DLL umax	1 à 99
			slotTime	USINT	DLL slotTime	dans des impulsions de 1 µs
			blanking	USINT	DLL blanking	dans des impulsions de 1,6 µs
			gb_start	USINT	DLL gb_start	dans des impulsions de 10 µs
			gb_center	USINT	DLL gb_center	dans des impulsions de 10 µs
			reserved	UINT	réservé	
			modulus	USINT	DLL modulus	127 requis
			gb_prestart	USINT	DLL gb_prestart	dans des impulsions de 10 µs
			TUI	STRUCT de 22 octets	Keeper TUI	voir 7.2.3.3
			unique_ID	UDINT	Keeper CRC	voir 7.2.3.4
			status_flag	UINT	TUI flag	voir 7.2.3.5
			réservé	USINT [16]	réservé	
0x81	obligatoire	Get	current_link_config	STRUCT de 34 octets	Paramètres de configuration de liaison en cours	identique à l'attribut 0x80 voir 6.9.2
0x82	obligatoire	Get/ Get_and _Clear	diagnostic_counters	STRUCT de 42 octets	compteurs de diagnostic	voir 7.2.3.7
			buffer_errors	UINT	compteur d'événements de la mémoire tampon	voir 7.2.3.8
			error_log	SWORD[8]	journalisation DLPDU erronée	voir 7.2.3.9
			event_counters	STRUCT de 32 octets	compteurs de diagnostic	voir 7.2.3.10
			good_frames_transmitted	SWORD[3]	DLPDU correctes transmises (LSB en premier)	
			good_frames_received	SWORD[3]	DLPDU correctes reçues (LSB en premier)	

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
			selected_channel_frame_errors	USINT	erreurs de tramage détectées sur le canal de réception actif	
			channel_A_frame_errors	USINT	erreurs de tramage détectées sur le canal A	
			channel_B_frame_errors	USINT	erreurs de tramage détectées sur le canal B	
			aborted_frames_transmitted	USINT	DLPDU annulées pendant la transmission (dépassement négatif de capacité de transmission)	
			highwaters	USINT	dépassement négatif de capacité de transmission LLC et dépassement de capacité de réception LLC	
			NUT_overloads	USINT	aucune durée planifiée dans la NUT (toutes les durées utilisées pour les transmissions planifiées)	
			slot_overloads	USINT	plus de données planifiées mises en file d'attente pour une NUT que le nombre admis par le paramètre sched_max_frame	
			blockages	USINT	une seule taille de Lpacket dépasse le paramètre de trame sched_max_	
			non_concurrence	USINT	au moins deux nœuds n'ont pas pu convenir de celui qui va assurer la transmission	
			aborted_frames_received	USINT	DLPDU incomplètes reçues	
			lonely_counter	USINT	nombre de fois que rien n'a été entendu sur le réseau pendant au moins 8 NUT	
			duplicate_node	USINT	DLPDU reçue d'un nœud avec un MAC ID de nœud local	
			noise_hits	USINT	bruit détecté ayant verrouillé le modem rx PLL	
			collisions	USINT	données Rx entendues juste au moment de la transmission	

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
			mod_MAC_ID	USINT	MAC ID du nœud modérateur en cours	
			non_lowman_mods	USINT	DLPDU modérateurs entendus depuis des nœuds non lowman	
			rogue_count	USINT	événements suspects entendus	
			unheard_moderator	USINT	DLPDU entendues, mais pas les modérateurs	
			vendor_specific	USINT	spécifique au fournisseur	
			réservé	SWORD[4]	réservé	
			vendor_specific	USINT	spécifique au fournisseur	
			vendor_specific	USINT	spécifique au fournisseur	
			réservé	SWORD	réservé	
0x83	obligatoire	Get	station_status	STRUCT de 6 octets	état de la station	voir 7.2.3.11
			smac_ver	USINT	mise en œuvre MAC	
			vendor_specific	SWORD[4]	spécifique au fournisseur	
			channel_state	SWORD	DEL d'état du canal, avertissement de redondance et bits de canal actif	voir 7.2.3.12
0x84	Get obligatoire, Set-(une seule fois) Facultatif	Get/Set	MAC_ID	STRUCT de 4 octets	commutateur MAC ID et paramètres en cours	
			MAC_ID_current	USINT	MAC ID en cours	Plage comprise entre 0 et 99. Voir 7.2.3.13.
			MAC_ID_switches	USINT	paramètres de commutation MAC ID	0 à 99, 0xFF. Voir 7.2.3.14
			MAC_ID_changed	BOOL	commutateurs MAC ID modifiés depuis la réinitialisation	voir 7.2.3.15
			Réservé	USINT	réservé	
0x85	Facultatif	Get/Set	Sched_max_frame	STRUCT de 2 octets	limite de données planifiée	
			Sched_max_frame	USINT	DLPDU maximal planifié	en paires d'octets. Voir 7.2.3.16
			Réservé	USINT	réservé	
0x86	Obligatoire	Get	error_log	STRUCT de 10 octets	Comptes d'erreur de la mémoire tampon du microprogramme du pilote et liste des nœuds gênants	Voir 7.2.3.17
			buffer_errors	UINT	compteur d'événements de la mémoire tampon	

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
			error_log	SWORD[8]	journalisation DLPDU erronée	
0x87	Facultatif	Get/ Get_and _Clear	extended_diagnostic_counters	STRUCT de 264 octets	compteurs de diagnostics supplémentaires	
			unsched_transmitted	UDINT	nombre de Lpackets non planifiés transmis	
			sched_highwater	UDINT	Nombre max. de Lpackets partagés/non planifiés dans la file d'attente de transmission	
			sched_data	128 STRUCTS de 2 octets (256 octets)	informations de planification	
			words_in_use	USINT	nombre de mots planifiés utilisés par ce nœud pendant la NUT	
			Lpkt_in_use	USINT	nombre de Lpackets planifiés utilisés par ce nœud pendant la NUT	
0x88	Facultatif	Get	Active_node_table	MATRICE de 32 octets	un bit pour chaque MAC ID 1 = nœud présent 0 = nœud manquant	voir 7.2.3.18
0x89	Facultatif	Get	New_node_table	MATRICE de 32 octets	un bit pour chaque MAC ID 1 = le nœud a rejoint la liaison récemment 0 = le nœud n'a pas rejoint la liaison récemment	voir 7.2.3.19
où LSB = octet de poids faible.						

7.2.3.2 pending_link_config

Les valeurs de l'attribut permettent de lire les paramètres de liaison en attente. Cet attribut doit être modifié sur la base de la fonction de gestion de station appelée modification de paramètre synchronisé et ne doit pas être défini par l'intermédiaire des services de l'objet ControlNet.

7.2.3.3 Keeper TUI

L'objet ControlNet doit conserver des exemplaires distincts du sous-attribut TUI pour les attributs pending_link_config et current_link_config. Les données TUI de l'attribut pending_link_config doivent être modifiées à la réception d'une balise fixe 0x81. Les données TUI de l'attribut current_link_config doivent être modifiées à l'issue d'une modification de paramètre synchronisé.

7.2.3.4 unique_ID

Le champ unique_ID doit être composé d'une valeur CRC de 32 bits calculée en fonction du contenu de la table de configuration de réseau gérée par l'objet Keeper. Cette valeur doit être copiée à partir du Lpacket 0x81 à balise fixe.

7.2.3.5 status_flag

Le champ status_flag du sous-attribut TUI doit être composé d'une valeur de 16 bits. Les bits non utilisés doivent être réservés et définis sur zéro.

Comme le montre le Tableau 15, le bit d'état de l'objet Keeper (bit 4) doit indiquer si ce nœud a déjà entendu un TUI provenant d'un objet Keeper maître. L'objet ControlNet doit annuler ce bit lors de l'initialisation. Étant donné qu'une valeur est attribuée au bit de tous les Lpackets TUI envoyés par l'objet Keeper maître, l'objet ControlNet ne doit pas manipuler ce bit.

La réception des Lpackets TUI doit être activée à la mise sous tension à l'aide du service DLL_enable_fixed_request(id, 0x84). Le champ d'état des Lpackets TUI doit être vérifié. Si le bit 3 = 0, le bit 4 = 1 et le bit 5 = 1, le Lpacket TUI doit être copié dans l'attribut 0x81 et la réception TUI être désactivée à l'aide de DLL_disable_fixed_request(id, 0x84). Les bits 0 et 1 du champ d'état TUI doivent être utilisés pour remplacer les paramètres de redondance de réseau par défaut du nœud. Ils doivent indiquer si le réseau est en cours de fonctionnement en mode de canal unique ou de canal redondant. Seules les informations de redondance du réseau provenant des Lpackets TUI dont les bits configurés Keeper State et Keeper sont définis doivent être utilisées.

Tableau 15 – Bits de balise d'état TUI

Bit	Signification
0 - 1	Redondance du réseau Bit 1 = 0, bit 0 = 0; Combinaison illégale Bit 1 = 0, bit 0 = 1; Canal B uniquement Bit 1 = 1, bit 0 = 0; Canal A uniquement Bit 1 = 1, bit 0 = 1; Redondant
2	Bit de gestion des ressources réseau 0 = Ressources réseau non gérées 1 = Ressources réseau gérées pour l'objet identifié dans le TUI
3	Bit de modification en cours du réseau 0 = Pas de modification de réseau en cours 1 = Modification de réseau en cours
4	Bit d'état de l'objet Keeper 0 = Le TUI transmis par l'objet Keeper n'est pas à l'état MAITRE 1 = Le TUI transmis par l'objet Keeper est à l'état MAITRE
5	Bit configuré de l'objet Keeper 0 = Aucun objet Keeper maître entendu depuis l'entrée dans la liaison 1 = Objet Keeper maître entendu depuis l'entrée dans la liaison
6 - 15	réservé (valeur 0 attribuée)

7.2.3.6 current_link_config

Les valeurs de l'attribut current_link_config correspondent aux paramètres de liaison actuellement utilisés. Le format de cet attribut doit être identique à celui de l'attribut pending_link_config.

7.2.3.7 diagnostic_counters

L'attribut `diagnostic_counters` doit gérer les comptages d'événements dans les couches physiques et de données.

NOTE L'attribut `diagnostic_counters` peut être lu sans affecter les valeurs grâce à l'une des demandes de service `Get_Attribute`. Les informations peuvent être lues et les valeurs réinitialisées à l'aide de la demande de service `Get_and_Clear`. Voir 7.2.5.

Les informations d'événement gérées par le microprogramme doivent également prendre la valeur `get` (mais pas `clear`) grâce à l'attribut `error_log`. Cela pour des raisons d'efficacité. Les informations gérées par le microprogramme doivent être immédiatement et directement accessibles.

7.2.3.8 buffer_errors

La valeur du sous-attribut `buffer_errors` doit être incrémentée à chaque événement de dépassement de capacité de réception et dépassement de capacité de transmission. Si une mise en œuvre ne dépasse ni ne fait jamais l'objet d'un dépassement négatif de ses capacités, une valeur nulle doit être attribuée à `buffer_errors`.

7.2.3.9 error_log

Si `DLL_event_indication` renvoie une valeur de `DLL_EV_badFrame`, le sous-attribut `error_log` enregistre les huit dernières indications de ce type. Le paramètre `mac_id` de cette indication doit être enregistré. L'ordre des MAC ID doit commencer par l'erreur la plus récente et se terminer par l'erreur la plus ancienne. Les valeurs nulles représentent les positions non utilisées dans le journal des erreurs.

7.2.3.10 event_counters

Le sous-attribut `event_counters` doit contenir les comptages d'événements de liaison. Le compteur doit s'arrêter lorsque les valeurs dépassent leurs limites.

7.2.3.11 station_status

Cet attribut doit uniquement prendre la valeur `get`.

7.2.3.12 channel_state

L'octet `channel_state` doit représenter l'état en cours des voyants d'état du réseau, s'ils sont mis en place. Les bits de l'octet `channel_state` sont présentés dans le Tableau 16.

NOTE Les voyants d'état du réseau sont présentés dans l'Annexe A.

Tableau 16 – Bits d'état du canal

Bits	Description
0,1,2	État de la DEL du canal A 0 = Inactive 1 = Vert fixe 2 = Vert clignotant 3 = Rouge clignotant 4 = Rouge-vert clignotant 5 = Chenillard Rouge 6 = Chenillard rouge-vert 7 = Rouge fixe
3,4,5	État de la DEL du canal B Idem DEL du canal A.
6	Avertissement de redondance – Le contrôleur ne peut pas utiliser le canal inactif d'une configuration redondante. 0 = normal (pas d'avertissement) 1 = avertissement
7	Canal actif – Indique lequel des deux canaux le récepteur en cours est en train d'écouter. 0 = Canal B 1 = Canal A

7.2.3.13 MAC_ID_current

La valeur MAC_ID_current doit être accessible à l'aide des services Get_Attribute. Le paramétrage doit être requis sur les dispositifs sans commutateur d'adresse. Seul le premier ensemble de valeurs MAC_ID_current après la réinitialisation d'un l'objet ControlNet doit être accepté. Le paramétrage doit être facultatif sur les dispositifs dotés de commutateurs d'adresse.

La plage de valeurs admises de MAC ID doit être comprise entre 1 et 99.

7.2.3.14 MAC_ID_switches

MAC_ID_switches doit prendre la valeur get et pas set sur tous les dispositifs. La valeur renvoyée doit être 0xFF pour les nœuds à adresse automatique et les nœuds sans commutateur d'adresse.

7.2.3.15 MAC_ID_changed

La valeur MAC_ID_changed doit être « clear » lorsque le dispositif est réinitialisé et « set » lorsque le dispositif détecte une modification des commutateurs d'adresse. Une fois définie, elle reste tant que le dispositif n'est pas réinitialisé, même si les commutateurs d'adresse reprennent leurs paramètres d'origine. La fréquence de vérification des commutateurs d'adresse doit dépendre du dispositif.

En cas de détection d'une modification des commutateurs d'adresse réseau, une valeur doit être attribuée à cette balise et une erreur mineure signalée à l'hôte.

7.2.3.16 sched_max_frame

L'attribut sched_max_frame doit être utilisé pour limiter la taille maximale des transmissions planifiées du nœud. Une valeur nulle doit indiquer qu'un nœud ne doit pas transmettre pendant la partie planifiée de la NUT.

7.2.3.17 error_log

L'attribut d'instance 0x86 doit être une copie du sous-attribut error_log contenu dans l'attribut d'instance 0x82.

NOTE Les informations de cet attribut sont une copie des informations disponibles dans l'attribut diagnostic_counters. Cette copie peut uniquement prendre la valeur get. Les valeurs de cet attribut peuvent être annulées uniquement à l'aide de la demande de service Get_and_Clear sur l'attribut diagnostic_counters. Voir 7.2.5 pour plus de détails.

Cette copie d'error_log doit uniquement être accessible par l'intermédiaire des services Get_Attribute. Cet attribut doit uniquement être annulé si la copie principale dans 0x82 l'est.

7.2.3.18 Table des nœuds actifs (attribut d'instance = 0x88)

L'attribut d'instance 0x88 de l'objet ControlNet doit être composé d'une matrice de 256 bits (un par MAC ID). Le bit de poids faible doit correspondre à MAC ID = 0. Le bit de poids fort doit correspondre à MAC ID = 255. Le bit d'un MAC ID particulier doit être défini (1) pour un nœud dans la seconde de la transmission de nœud sur la liaison. Le bit doit être nul (0) dans les 20 s du nœud qui quitte la liaison. Un nœud doit être considéré comme ayant quitté la liaison s'il manque trois opportunités de transmission consécutives.

7.2.3.19 Nouvelle table de nœuds (attribut d'instance = 0x89)

L'attribut d'instance 0x89 de l'objet ControlNet doit être composé d'une matrice de 256 bits (un par MAC ID). Le bit de poids faible doit correspondre à MAC ID = 0. Le bit de poids fort doit correspondre à MAC ID = 255. Le bit d'un MAC ID particulier doit être défini (1) pour un nœud dans la seconde d'une transmission de nœud sur la liaison. Le bit doit être nul (0) entre 10 s et 20 s après que le nœud a rejoint la liaison.

7.2.4 Services communs

7.2.4.1 Généralités

L'objet ControlNet doit prendre en charge les services communs spécifiés dans le Tableau 17.

Tableau 17 – Services communs de l'objet ControlNet

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service
	Classe	Instance		
0x05	N/A	Requise	Reset	Renvoie l'objet ControlNet à l'état de mise sous tension
0x10	N/A	Sous condition	Set_Attribute_Single	Définit les informations de configuration spécifiques au réseau ou au nœud
0x0E	Requise	Requise	Get_Attribute_Single	Obtient les informations de configuration spécifiques au réseau ou au nœud
0x01	N/A	Facultatif	Get_Attribute_All	Obtient les informations de configuration spécifiques au réseau et au nœud
0x02	N/A	Facultatif	Set_Attribute_All	Définit les informations de configuration spécifiques au réseau et au nœud
0x03	N/A	Facultatif	Get_Attribute_List	Obtient les informations de configuration spécifiques au réseau et/ou au nœud
0x04	N/A	Facultatif	Set_Attribute_List	Définit les informations de configuration spécifiques au réseau et/ou au nœud

7.2.4.2 Service Reset

Le service Reset doit réinitialiser le Moniteur de connexion de réseau (NAM) afin de surveiller l'activité de liaison, puis tente de rejoindre la liaison. Les commutateurs MAC ID ne doivent pas être réévalués en raison de la réinitialisation.

Il n'est pas utile que l'objet ControlNet réponde à une demande Reset avant de réellement lancer une opération de réinitialisation.

7.2.4.3 Set_Attribute_Single

Si un attribut a été mis en œuvre comme étant définissable, le service Set_Attribute_Single doit être requis.

7.2.4.4 Réponse Get_Attribute_All

La partie de données de réponse spécifique à l'objet/au service d'une réponse Get_Attribute réussie est présentée dans les attributs d'instance de la table d'objets ControlNet. La taille de cette réponse dépend des instances d'attribut en cours d'accès.

Compte tenu de la taille importante de la réponse Get_Attribute_All, il n'est pas utile que les dispositifs aux ressources limitées prennent en charge ce service.

La partie de données d'une demande de service Get_Attribute_All doit contenir les attributs suivants dans l'ordre suivant:

0x81 - current_net_config

0x82 - diagnostic_counters

0x83 - station_status

0x84 - MAC_ID

0x86 - error_log

7.2.5 Services spécifiques à la classe

7.2.5.1 Généralités

L'objet ControlNet doit prendre en charge les services spécifiques à la classe spécifiés dans le Tableau 18.

Tableau 18 – Services spécifiques à la classe de l'objet ControlNet

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service
	Classe	Instance		
0x4C	N/A	Requise	Get_and_Clear	Obtient, puis annule les compteurs de diagnostic
0x4D	N/A	Requise	Enter_Listen_Only	Force le nœud à entrer et rester en mode d'écoute seulement jusqu'à la réinitialisation
0x4E	N/A	Requise (Voir 0)	Where_am_I	Détermine le MAC ID du dispositif auquel est associé ce nœud par l'intermédiaire du NAP
0x4F	N/A	Sous condition	Auto_Address	Requis dans tous les nœuds d'adresse automatique. Sélectionner un nouveau MAC ID à l'aide de la séquence d'adresse automatique du NAM

7.2.5.2 Service Get_and_Clear

Ce service spécifique à la classe doit appeler une réponse identique au service Get_Attribute_Single. Toutefois, après la génération du Lpacket de réponse, la valeur de l'attribut doit être nulle. Seuls les attributs d'instance 0x82 et 0x87 doivent prendre en charge ce service.

7.2.5.3 Service Enter_Listen_Only

Le service Enter_Listen_Only doit inciter cet objet à envoyer un `DLL_listen_only_request(FALSE)` à la DLL associée à cette instance de l'objet ControlNet. Il n'est pas utile que ce service génère une réponse étant donné que le nœud peut s'arrêter de transmettre avant l'envoi de la réponse.

NOTE 1 Il convient que le nœud reste à l'état d'écoute seulement tant que la demande de réinitialisation n'est pas envoyée à l'objet ControlNet ou Identity.

NOTE 2 Le service Enter_Listen_Only peut être utilisé pour les besoins du diagnostic afin de forcer les nœuds à diminuer la liaison tant qu'ils n'ont pas été spécifiquement invités à la rejoindre. Cela permet de retirer effectivement un nœud de la liaison sans déconnecter physiquement le câble réseau.

7.2.5.4 Service Where_am_I

Tous les nœuds transitoires doivent mettre en œuvre le service Where_am_I. Ce service doit déterminer le MAC ID du nœud auquel le nœud transitoire est connecté par l'intermédiaire de son port d'accès au réseau (NAP). Le service peut être mis en œuvre à l'aide de la fonctionnalité de serveur WAMI de l'entité de gestion de station.

NOTE Un nœud transitoire peut utiliser ce service afin d'identifier le MAC ID du nœud au NAP duquel il est connecté.

La réponse Where_am_I doit être un USINT unique contenant le MAC ID du nœud permanent auquel le nœud transitoire est connecté. Si l'objet ControlNet n'est pas en mesure de terminer la requête WAMI, il doit renvoyer un MAC ID de 255.

7.2.5.5 Service Auto_Address

Pour les nœuds qui prennent automatiquement en charge la sélection d'un MAC ID, ce service doit réinitialiser le NAM afin de surveiller l'activité de liaison, puis tenter de joindre la liaison avec un nouveau MAC ID, si possible. Les nœuds qui ne prennent pas automatiquement en charge la sélection d'un MAC ID ne doivent pas mettre en œuvre ce service.

Il n'est pas utile que le service Auto_Address réponde avant la recherche d'un nouveau MAC_ID.

7.2.6 Comportement

7.2.6.1 Effet de DLL_tminus_zero_indication

A la réception de cette indication, l'objet doit copier le contenu de l'attribut d'instance 0x80 (pending_link_config) dans l'attribut d'instance 0x81 (current_link_config). Une copie interne de l'attribut d'instance 0x80 doit être conservée, même si la mise en œuvre ne l'a pas rendue accessible par l'intermédiaire des services Get_Attribute.

7.2.6.2 Code de duplication

Si un `DLL_event_indication(DLL_EV_dupNode)` est reçu de la couche de liaison de données, l'objet ControlNet doit forcer la DLL à passer à l'état d'écoute seulement. L'objet ControlNet doit conserver cet état tant qu'un service de réinitialisation n'a pas été reçu. Une réinitialisation de l'objet Identity doit également mettre fin à l'état d'écoute seulement.

7.2.6.3 Redondance

Une valeur doit être attribuée au paramètre de redondance du réseau d'un nœud au démarrage afin d'obtenir les meilleures capacités du dispositif. (Le canal A des dispositifs à canal unique, les deux canaux des dispositifs redondants du réseau). Le paramètre de redondance de mise sous tension doit apparaître dans le TUI de l'attribut d'instance 0x82.

L'objet Keeper doit mettre à jour le paramètre de redondance du réseau d'un nœud avec le paramètre en cours de l'une des deux manières suivantes:

- Le nœud reçoit un Lpacket TUI du réseau contenant l'objet Keeper maître, les bits de l'objet Keeper configuré et le bit de variation nette en cours réinitialisé dans le mot Status_Flags;
- Le nœud reçoit un Lpacket 0x81 à balise fixe.

Dans tous les cas, les données TUI reçues doivent être copiées dans l'attribut 0x80.

Si un nœud à canal unique rejoint une liaison redondante, il doit se comporter comme un nœud redondant avec un canal défaillant.

7.2.7 Voyant d'état du module

Le voyant d'état du module, s'il est mis en œuvre, doit apporter les indications d'état suivantes:

- a) vert fixe lorsque les connexions sont actives dans l'état *en ligne*;
- b) vert clignotant/désactivé à l'état de *vérification de l'écoute du modérateur uniquement*;
- c) vert clignotant/désactivé lorsque des conditions suspectes sont détectées;
- d) rouge clignotant/désactivé en cas d'erreur réparable;
- e) rouge clignotant/désactivé à l'état *DUP_LISTEN_ONLY*;
- f) rouge fixe en cas d'erreur irréparable.

7.3 Objet Keeper

7.3.1 Présentation

L'objet Keeper doit détenir tous les attributs de liaison de tous les dispositifs d'une liaison et doit être chargé de les distribuer aux dispositifs de manière ordonnée. Il détient également (pour le logiciel de planification de liaison) un exemplaire des données de l'auteur de la connexion de tous les dispositifs à l'origine de la connexion utilisant un réseau. Si plusieurs objets Keeper sont présents sur une liaison, ils procèdent à des négociations afin de déterminer celui qui va devenir maître et ceux qui vont assurer les fonctions de sauvegarde. Le Keeper maître est celui qui distribue activement les attributs aux nœuds du réseau. Un objet Keeper de sauvegarde est celui qui surveille l'activité réseau associée au Keeper et peut assurer le rôle d'objet Keeper maître en cas de défaillance de celui-là.

L'objet Keeper contient également un support pour l'obtention, la gestion et la délivrance de la ressource réseau, sémaphore réseau utilisé pour éliminer les conflits lors de la modification des données d'attribut détenues par le/les objet(s) Keeper sur une liaison.

7.3.2 Historique de révision

L'historique de révision de l'objet Keeper est décrit dans le Tableau 19.

Tableau 19 – Historique de révision de l'objet Keeper

Révision de classe	Description des modifications
01	Remise initiale
02	Remise pour la série CEI 61158

7.3.3 Attributs de classe

L'objet Keeper doit prendre en charge les attributs de classe spécifiés dans le Tableau 20.

Tableau 20 – Attributs de classe de l'objet Keeper

Numéro	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x1	Requise	Get	Revision	UINT	Révision de cet objet	2 (révision 2) (Voir Note 2)
NOTE 1 Les dispositifs Revision 1 Keeper peuvent fonctionner uniquement si MAC ID = 1. Ces dispositifs exécutent les fonctions de l'objet Keeper conformément à la présente spécification tant qu'un autre objet Keeper n'est pas présent sur le réseau. Avec un autre MAC ID, les dispositifs contenant les objets Keeper - révision 1 fonctionnent comme si aucun objet n'était présent. NOTE 2 Les dispositifs Revision 2 Keepers fonctionnent avec n'importe quelle valeur MAC ID légale. Ces dispositifs exécutent les fonctions de l'objet Keeper conformément à la présente spécification en présence d'autres objets Revision 2 Keeper sur le réseau.						

7.3.4 Attributs d'instance

7.3.4.1 Généralités

L'objet Keeper doit prendre en charge les attributs d'instance spécifiés dans le Tableau 21. Une seule instance de l'objet Keeper est admise. Cette instance doit être sensible au port sur lequel la demande a été reçue. Cela est particulièrement important en présence d'un routeur sur lequel une instance Keeper reçoit des demandes provenant de deux liaisons.

Tableau 21 – Attributs d'instance de l'objet Keeper

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x00	Requise	Get	état	STRUCT de 2 octets	État de l'objet Keeper	
			état	USINT	État de fonctionnement de l'objet Keeper en cours	
			réservé	USINT	Réservé pour l'alignement des données	
0x01 à 0x63	Requise	Get/Set	port_status	STRUCT de 10 octets	État du port des nœuds 1 à 99	
			port_status	UINT	État du port	
			ID	STRUCT de 8 octets	Identification du type de nœud	
			vendor	UINT	Code fournisseur	
			type	UINT	Type de produit	
			code	UINT	Code produit	
			major	USINT	Révision majeure	
			minor	USINT	Révision mineure	
0x64 – 0xFE	Réservé					Réservé pour une extension ultérieure
0xFF	Requise	Get/Set	net_config	STRUCT de 12 octets	Paramètres de réseau en cours	
			NUT	UINT	Durée de mise à jour du réseau	Dans des impulsions de 10 µs
			smax	USINT	ID nœud max planifié	0 à 99
			umax	USINT	ID nœud max non planifié	0 à 99
			Slot_Time	USINT	Intervalle	Dans des impulsions de 1 µs
			Blank_Time	USINT	Temps de blocage	En octets
			Gb_Start	USINT	Début de la bande de garde	Dans des impulsions de 10 µs
			Gb_Center	USINT	Centre de la bande de garde	Dans des impulsions de 10 µs
			Réservé	UINT	Réservé pour l'alignement des données	
			Int_Cnt mod	USINT	Module de comptage interne (longueur de macrocycle)	127
			Gb_Prestart	USINT	Démarrage préalable de la bande de garde	Dans des impulsions de 10 µs
0x100	Requise	Get/Set	name	UINT [33]	nom actuel de cette liaison	32 caractères unicode et un NULL unicode
0x101	Requise	Get/Set	RT_TUI	STRUCT de 22 octets	Identifiant unique de table	
			unique_ID	UDINT	attribut CRC	
			status_flag	UINT	Bits de balise TUI	Voir 7.2.3.3

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
			réservé	USINT [16]	réservé à l'admission d'un format commun avec l'attribut 0x102	
0x102	Requise	Get	Link_TUI	STRUCT de 22 octets	Identifiant unique de table (liaison en cours uniquement)	Requise
			unique_ID	UDINT	attribut CRC	
			status_flag	UINT	Bits de balise TUI	Voir 7.2.3.3
			réservé	USINT	Réservé pour l'alignement des données	Un MAC ID légal
			Net_Resource_Vendor_Id	UINT	ID fournisseur de l'objet détenant le Net Resource	
			Net_Resource_Serial_Number	UDINT	Numéro de série de l'objet détenant le Net Resource	
			Net_Resource_Class	UDINT	Classe de l'objet détenant le Net Resource	
			Net_Resource_Instance	UDINT	Instance de l'objet détenant le Net Resource	
			réservé	USINT	Réservé pour l'alignement des données	

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x103	Requise	Get/Set	Cable_Config	STRUCT de 100 octets au maximum	Configuration de câble en cours	
			Id	USINT	Valeur de l'ID	Toujours 0xFF
			Num_Elements	USINT	Nombre d'éléments de configuration de câble dans une configuration de réseau	Compris entre 1 et 24
			Propagation_time	UINT	Nombre d'impulsions 100 ns	
			Élément physique	Élément phy [24]		
			Phy_element	STRUCT de 4 octets		
			Vendor_id	UINT	Code fournisseur	
			Product_code	USINT	Code produit	
			How_many	USINT		
0x104	Requise	Get/Set	CO_summary	STRUCT de 204 octets	Informations générales relatives à l'attribut co_data	
			data_size	UINT	Taille de l'attribut co_data	En mots
			connection_info_revision	USINT		Utilisé par les outils de programmation 0 = Format 1 1 = Format 2 2 – 255 réservés
			tool_keeper_revision	USINT	Règle d'analyse syntaxique co_data de niveau le plus élevé prise en charge	Niveau pris en charge ≤ révision de l'objet Keeper 0 = Format 1 1 – 255 réservés Voir 7.3.6.7
			Offsets	UINT [100]	Matrice de décalages de mot dans l'attribut co_data, par MAC ID	En mots, 0xFFFF = aucune donnée pour ce nœud 0 MAC ID selon l'indice
0x105	Requise	Fragment Get/Set	CO_data	STRUCT de 14 988 octets au maximum	Informations sur l'auteur de la connexion	Format 1 de l'outil Keeper Revision
			Branch	STRUCT de taille variable	Informations relatives à tous les routeurs ou auteurs de la connexion sur cette liaison	
			num_devices	UINT	Nombre de dispositifs sur cette liaison	
			device	STRUCT de taille variable	détails de ce dispositif	

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
			type	USINT	types de dispositif	1 = routeur 2 = auteur de la connexion
			Path_Size	USINT	Taille du chemin vers le nœud	En mots de 16 bits
			Path	MATRICE d'UINT	Chemin vers le dispositif	
			CO_data	STRUCT de taille variable	Détails des données CO	Présent uniquement pour les dispositifs CO (type = 2)
			COP	UDINT	Mot de passe CO, tel qu'indiqué au CO à la fin d'une session planifiée (voir 7.4.5.8)	
			Size	UINT	Taille de la connexion	En mots de 16 bits
			Connexion	STRUCT de taille variable	Données de connexion	Révision d'informations de connexion au format spécifié par connexion_info_revision de l'attribut 0x104 Une par connexion pour ce dispositif CO. Voir 7.3.4.7

7.3.4.2 Définitions de l'état de fonctionnement

Les valeurs de l'état de fonctionnement de l'objet Keeper doivent être telles que définies dans le Tableau 22.

Tableau 22 – Définitions de l'état de fonctionnement de l'objet Keeper

Valeur	Description
0	Mise sous tension – Pas en ligne
1	Mise sous tension – Attente TUI
2	Mise sous tension – Interrogation TUI
3	Sauvegarde
4	Vérification du maître
5	Maître
6	Échec de la sauvegarde
7	Échec de la vérification du maître
8	Échec du maître
9	Variation nette - Sauvegarde
10	Variation nette - Maître
11	Variation nette – Échec de la sauvegarde
12	Variation nette – Échec du maître

7.3.4.3 Définitions de bit de balise d'état du port

Les valeurs de la balise d'état du port doivent être telles que définies dans le Tableau 23.

Tableau 23 – Définitions de bit de balise d'état du port

Bit	Signification
0	Le nœud accepte le bit de connexion planifiée 0 = Le nœud n'accepte pas les connexions planifiées 1 = Le nœud accepte les connexions planifiées
1 – 15	réservé (valeur 0 attribuée)

7.3.4.4 status_flag

Le bit configuré de l'objet Keeper annulé dans le mot d'état TUI doit identifier un objet Keeper doté d'une configuration non valide ou d'une mémoire effacée.

Conformément au Tableau 24, le champ status_flag du sous-attribut TUI doit être composé d'une valeur de 16 bits. Les bits de balise non utilisés doivent être réservés et définis sur 0.

Le bit de gestion de ressource réseau doit uniquement être utilisé par l'objet Keeper.

Le bit de l'objet Keeper maître doit indiquer si ce nœud a déjà entendu ou pas un objet Keeper maître. L'objet ControlNet doit annuler ce bit lors de l'initialisation. Une valeur doit être attribuée à ce bit de tous les TUI envoyés par l'objet Keeper maître. Si un Lpacket TUI est reçu et qu'aucune valeur n'a été attribuée au bit de l'objet Keeper maître, le Lpacket doit être conservé comme étant un TUI reçu, mais les informations de redondance ne doivent pas être utilisées.

Les bits de redondance doivent être utilisés pour remplacer les paramètres de redondance par défaut pour le nœud. Ils doivent indiquer si la liaison est en cours de fonctionnement en mode de canal unique ou redondant. Les informations de redondance ne doivent pas être utilisées si un Lpacket TUI est reçu et que le bit de l'objet Keeper maître est absent.

Le bit de variation nette en cours doit indiquer qu'une séquence de modification de réseau synchronisée est en cours et que de nouveaux nœuds doivent retarder leur mise en ligne tant que la séquence de modification n'est pas terminée et que le bit n'est pas supprimé. Ce bit doit être annulé en cas de dépassement de délai de ressource réseau ou si la ressource réseau est libérée.

Tableau 24 – Bits de balise d'état TUI

Bit	Signification	Utilisé dans:		
		RT_TUI	Network_TUI	ControlNet object TUI
0 - 1	Redondance du réseau Bit 1 = 0, bit 0 = 0; Combinaison illégale Bit 1 = 0, bit 0 = 1; Canal B uniquement Bit 1 = 1, bit 0 = 0; Canal A uniquement Bit 1 = 1, bit 0 = 1; Redondant	X	X	X
2	Bit de gestion des ressources réseau 0 = Ressources réseau non gérées 1 = Ressources réseau gérées pour l'objet identifié dans le TUI		X	
3	Bit de modification en cours du réseau 0 = Pas de modification de réseau en cours 1 = Modification de réseau en cours	X	X	X
4	Bit d'état de l'objet Keeper 0 = Le TUI transmis par l'objet Keeper n'est pas à l'état MAITRE 1 = Le TUI transmis par l'objet Keeper est à l'état MAITRE		X	X
5	Bit configuré de l'objet Keeper 0 = Attributs Keeper non configurés (RT_TUI) 1 = Attributs Keeper configurés (RT_TUI)	X	X	X
6 - 15	réservé (valeur 0 attribuée)			

7.3.4.5 unique_ID

Le champ unique_ID de l'attribut TUI doit être composé d'une valeur de 32 bits calculée en fonction du contenu du tableau de l'attribut Keeper (autre l'attribut TUI) par l'outil de configuration logicielle. Les attributs suivants doivent être utilisés, dans l'ordre donné pour calculer l'attribut unique_ID TUI:

0x1 à 0x63 (paramètres de port des nœuds 1 à 99)

0xFF (paramètres de liaison)

0x100 (nom de liaison);

0x101 (RT TUI) bits de redondance uniquement;

0x103 (configuration de câble);

0x105 (données de mot de passe CO uniquement);

L'identifiant unique TUI doit être calculé comme suit:

— l'identifiant unique TUI doit être remis à zéro;

— Les attributs de données 0x1 à 0x63, l'attribut 0xFF, l'attribut 0x100, le paramètre de redondance (attribut 0x101, balise d'état RT_TUI, bits 0–1), 0x103 et chaque mot de passe CO (dans l'ordre indiqué) doivent être exécutés par l'intermédiaire du polynôme CRC.

Le polynôme utilisé pour calculer le CRC doit être

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + 1$$

(voir la Figure 17).


```

UDINT CRC(
    unsigned long CRC, // (r) the updated CRC
    SWORD* buffer,     // (i) the CRC to start with
    WORD size)         // (i) pointer to buffer of SWORD / octets
                    // (i) the number of octets in the buffer

# define KEEPER_CRC_POLY 0xEDB88320L
WORD octetIndex;
WORD bitIndex;
SWORD oneOctet;

// loop through the octets passed, including them in the CRC
for (octetIndex=0; octetIndex < size; octetIndex++)
{
    oneOctet = buffer[octetIndex];

// loop through the bits passed, including them in the CRC
for (bitIndex = 0; bitIndex < 8; bitIndex++)
{
    if ((oneOctet & 0x1) ^ (CRC & 0x1))
    {
        CRC = (CRC >> 1) ^ KEEPER_CRC_POLY;
    } else {
        CRC = (CRC >> 1);
    }
    oneOctet >>= 1;
}
}

// return the new CRC
return(CRC);
}

```

Figure 17 – Algorithme CRC de l'objet Keeper

7.3.4.6 Keeper MAC_ID

Il convient que le Keeper_MAC_ID soit le MAC ID de l'objet Keeper maître en cours diffusant le TUI. Si aucun TUI n'a été reçu au cours des 12 dernières NUT, une valeur de 0xFF doit être renvoyée

7.3.4.7 Connexion

Il existe différents formats de données de connexion. Chacun d'eux est décrit ci-dessous. L'ensemble des entrées de données de connexion doit être un nombre entier de mots.

Le format 1 ne précise pas la taille du chemin. Par conséquent, son chemin [] doit se présenter sous deux formes:

- segment de classe, segment d'instance, puis deux segments de point de connexion; ou
- un segment de symbole étendu ANSI.

```

// Format 1
USINT  O2T_Size      // in 16 bit words
UINT    O2T_Rpi      // in 1 ms ticks from 2 to 32 767
USINT  T2O_Size      // in 16 bit words
UINT    T2O_RPI      // bit 15 = connection point, 0 = point to point, 1 = multipoint
                    // bit 14 = API in ms from 2 to 32 767
USINT  O2T_Nui_Api   // The highest bit set indicates the API;
                    // bit 7 = 128*NUT, bit 6 = 64*NUT, etc.
                    // Remaining bits indicate the O2T_NUI. If API = 1, NUI = 0
USINT  T_Node        // Range 1 - 99
USINT  T2O_Nui_Api   // The highest bit set indicates the API;
                    // bit 7 = 128*NUT, bit 6 = 64*NUT, etc.
                    // Remaining bits indicate the O2T_NUI. If API = 1, NUI = 0
UINT    path[]       // forward open connection path from this

```

```
// Format 2
UINT  O2T_net_params // copied from forward open
UINT  T2O_net_params // copied from forward open
UINT  O2T_RPI         // RPI in floating point format
UINT  T2O_RPI         // RPI in floating point format
USINT  O2T_schedule   // from schedule segment in forward open
USINT  MAC_ID         // from schedule segment in forward open
USINT  T2O_schedule   // from schedule segment in forward open
USINT  path_size      // in 16-bit words
UINT  path[]          // forward open connection path from this
```

```
UDINT convert_to_10us_ticks (UINT RPI)
{
    return (RPI <= 0x4000) ? RPI :
           (RPI <= 0xE000) ? (RPI & 0x1FFF) + 0x2000 << (RPI>>13) - 1 :
           : (RPI & 0x0FFF) + 0x1000 << (RPI>>12 & 1) + 7;
}

UINT convert_to_Keeper_format (UDINT ticks)
{
    if (ticks < 0x00004000) return RPI;
    if (ticks < 0x00008000) return RPI>>1 & 0x1FFF | 0x4000; // 16384 - 32766 by 2
    if (ticks < 0x00010000) return RPI>>2 & 0x1FFF | 0x6000; // 32768 - 65532 by 4
    if (ticks < 0x00020000) return RPI>>3 & 0x1FFF | 0x8000; // 65536 - 131064 by 8
    if (ticks < 0x00040000) return RPI>>4 & 0x1FFF | 0xA000; // 131072 - 262128 by 16
    if (ticks < 0x00080000) return RPI>>5 & 0x1FFF | 0xC000; // 262144 - 524256 by 32
    if (ticks < 0x00100000) return RPI>>7 & 0x0FFF | 0xD000; // 524288 - 1048448 by 128
    if (ticks < 0x00200000) return RPI>>8 & 0x0FFF | 0xE000; // 1048576 - 2096896 by 256
    return 0xFFFF;
}
```

7.3.4.8 Attribute data

Un objet Keeper conserve la liaison et ses informations de configuration nœud dans une structure de données d'attribut. L'objet Keeper doit conserver deux exemplaires de la structure d'attribut: un exemplaire en attente dans la mémoire RAM et un exemplaire en cours dans la mémoire non volatile. Lorsque de nouveaux attributs sont définis dans l'objet Keeper, ils doivent l'être dans l'exemplaire en attente dans la mémoire RAM. L'exemplaire RAM doit être écrit dans l'exemplaire actuellement actif de la mémoire non volatile dès la réception d'une demande spéciale.

Les attributs doivent s'appuyer sur les règles suivantes pour l'obtention et la définition:

- Les demandes de service de définition définissent l'exemplaire en cours dans la mémoire RAM;
- Les demandes de service d'obtention obtiennent l'exemplaire en cours dans la mémoire non volatile.

L'objet Keeper doit conserver l'ensemble d'attributs spécifié dans le Tableau 25:

Tableau 25 – Attributs Keeper

Numéro d'attribut	Attributs Keeper
0x01 - 0x63	(paramètres de port des nœuds 1 à 99)
0xFF	(paramètres de liaison)
0x100	(nom de liaison)
0x101	(Keeper TUI)
0x103	(configuration de câble)
0x104	(informations de décalage de nœud dans les données CO)
0x105	(chemin CO et informations sur le mot de passe)

Toutes les définitions d'attribut Keeper supposent la présence de 99 nœuds au maximum sur la liaison. L'objet Keeper doit disposer d'une mémoire suffisante pour stocker les attributs CO_summary et CO_data.

Les exigences en matière de mémoire (en octets) des attributs Keeper sont spécifiées dans le Tableau 26.

Tableau 26 – Exigences en matière de mémoire (en octets) des attributs Keeper

Octets	Attributs Keeper
2	État de l'objet Keeper
990	Paramètres de port (99 nœuds de 10 octets chacun)
12	Paramètres de liaison
66	nom de liaison
22	TUI
100	Configuration de câble
204	Récapitulatif CO
14 988	Données CO (7 494 mots)
16 384	Total (Exactement 16 Ko)

La séquence d'événements requise pour mettre à jour les attributs Keeper correctement et en toute sécurité doit se présenter comme suit:

- obtenir la ressource réseau;
- lancer une demande de service Change_Start à l'objet Keeper;
- lancer la/les demande(s) de service Set_Attribute appropriée(s) afin de mettre à jour l'exemplaire en attente des attributs Keeper;
- lancer une demande de service Change_Complete à l'objet Keeper. Cette procédure doit initier une modification de paramètre synchronisé;
- libérer la ressource réseau.

7.3.5 Services communs

Les services communs Keeper doivent tous fonctionner avec un temps interne afin d'éviter qu'une condition d'erreur matérielle ne bloque indéfiniment les opérations. Les valeurs réelles de délai d'attente doivent dépendre du dispositif.

L'objet Keeper doit prendre en charge les services communs spécifiés dans le Tableau 27.

Tableau 27 – Services communs de l'objet Keeper

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service
	Classe	Instance		
0x03	N/A	Requise	Get_Attribute_List	Obtient les informations de configuration spécifiques au réseau et au nœud
0x04	N/A	Requise	Set_Attribute_List	Définit les informations de configuration spécifiques au réseau et au nœud
0x0E	Requise	Requise	Get_Attribute_Single	Obtient les informations de configuration spécifiques au réseau et au nœud
0x10	N/A	Requise	Set_Attribute_Single	Définit les informations de configuration spécifiques au réseau et au nœud

L'objet Keeper est en mesure de recevoir ces demandes de service par l'intermédiaire des balises fixes 83 et 88. S'il les reçoit par la balise fixe 83, la demande est traitée et fait l'objet d'une réponse indépendamment de l'état de l'objet Keeper. S'il les reçoit par la balise fixe 88, l'objet Keeper répond uniquement à la demande s'il est à l'état Maître, Variation nette - Maître, Échec du maître ou Variation nette – Échec du maître. Les nœuds Keeper se trouvant dans d'autres états ne doivent pas répondre à la demande.

7.3.6 Services spécifiques à la classe

7.3.6.1 Généralités

Tous les dispositifs ayant mis en œuvre l'objet Keeper doivent également mettre en œuvre le Keeper UCMM.

L'objet Keeper doit prendre en charge les services spécifiques à la classe suivants spécifiés dans le Tableau 28.

Tableau 28 – Services spécifiques à la classe de l'objet Keeper

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service	Paramètres(s)
	Classe	Instance			
0x4B	N/A	Requise	Obtain_Network_Resource (ONR)	Tentative d'obtention de la ressource réseau pour cette liaison. Si la tentative aboutit, maintenir pendant 60 s	UINT ID fournisseur UDINT numéro de série UDINT classe UDINT instance
0x4C	N/A	Requise	Hold_Network_Resource (HNR)	Continuer à maintenir la ressource réseau précédemment obtenue pour cette liaison pendant 60 s	UINT ID fournisseur UDINT numéro de série UDINT classe UDINT instance
0x4D	N/A	Requise	Release_Network_Resource (RNR)	Libérer la ressource réseau pour cette liaison	UINT ID fournisseur UDINT numéro de série UDINT classe UDINT instance
0x4E	N/A	Requise	Change_Start (CS)	Copier les attributs en cours de la mémoire non volatile dans les attributs en attente de la mémoire RAM. Entrer dans l'état de fonctionnement de <i>variation nette</i> approprié	<aucun>
0x4F	N/A	Requise	Change_Complete (CC)	Copier les données d'attribut en attente de la mémoire RAM dans la mémoire non volatile. Reprendre le fonctionnement normal de l'objet Keeper. Lancer une modification de réseau synchronisée si les attributs nets ont été modifiés. Si le nombre d'ensembles ne correspond pas, procéder à un traitement <i>change_abort</i>	Nombre d'« ensembles » à 16 bits exécutés depuis <i>Change_Start</i>
0x50	N/A	Requise	Change_Abort (CA)	Ignorer les modifications apportées aux données d'attribut en attente, libérer la ressource réseau, puis rétablir le fonctionnement normal de l'objet Keeper	<aucun>
0x51	N/A	Requise	Get_Signature (GS)	Obtenir la valeur de signature d'un auteur de connexion particulier (mot de passe CO)	Demande: taille de chemin (octet) (la taille est en mots) chemin Réponse: cop 32 bits
0x52	N/A	Requise	Get_Attribute_Fragment (GAF)	Obtenir une partie d'un attribut spécifiquement conçu pour être fragmenté et trop volumineux pour entrer dans un seul Lpacket. Utilisation réservée à l'attribut <i>co_data</i>	Taille UINT; //words Décalage UINT; //offset

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service	Paramètres(s)
	Classe	Instance			
0x53	N/A	Requise	Set_Attribute_Fragment (SAF)	Définir une partie d'un attribut spécifiquement conçu pour être fragmenté et trop volumineux pour entrer dans un seul Lpacket. Utilisation réservée à l'attribut CO_data	Taille UINT; //words Décalage UINT; //offset Données UINT []

Les codes d'erreur de ces services doivent être ceux spécifiés dans le Tableau 29, les codes de service étant les abréviations des noms de service spécifiés dans le Tableau 28.

Tableau 29 – Codes d'erreur du service

Code d'erreur	Signification	Code d'erreur utilisé par le service								
		ONR	HNR	RNR	CS	CC	CA	CS	GAF	SAF
0x02	resource_unavailable_err	X	X	X	X					
0x03	invalid_parameter_err					X		X	X	X
0x04	path_segment_err							X	X	X
0x05	path_dest_unknown_err	X	X	X	X	X	X	X	X	X
0x08	unimplemented_service_err	X	X	X	X	X	X	X	X	X
0x09	invalid_attr_err								X	X
0x0C	wrong_object_state_err					X	X			X
0x0D	already_exists_err	X			X					
0x0F	no_permission_err		X	X						
0x11	reply_too_large_err							X		
0x13	not_enough_data_err	X	X	X		X		X	X	X
0x14	undefined_attr_err		X							
0x15	too_much_data_err							X		X
0x16	nonexistent_object_err	X	X	X	X	X	X	X	X	X
0x26	invalid_path_size_err	X	X	X	X	X	X	X	X	X

7.3.6.2 Ressource réseau

La ressource réseau (NR) doit être un sémaphore de liaison permettant d'éliminer les conflits en cas de modification des informations d'attribut des objets Keeper.

En cours de fonctionnement, la ressource réseau doit apparaître sous la forme d'un bit de balise d'état et d'autres champs de données du Lpacket TUI qui doivent être diffusés de manière répétitive par l'objet Keeper maître.

Un dispositif souhaitant mettre à jour les informations d'attribut de l'objet Keeper doit en premier lieu soumettre une demande à l'objet Keeper maître afin d'obtenir la ressource réseau pour son compte (par l'intermédiaire d'un service Obtain_Network_Resource). Cette demande doit être diffusée étant donné que l'identité de l'objet Keeper maître n'est en général pas connue. Une fois obtenu, le dispositif doit régulièrement demander (toutes les 15 s) à l'objet Keeper maître de continuer à gérer la ressource réseau par l'intermédiaire d'une demande de service Hold_Network_Resource. L'objet Keeper maître doit gérer un temporisateur de 60 s, qui est démarré dès l'obtention de la ressource réseau, puis de

nouveau déclenché dès la réception d'une demande *Hold_Network_Resource* consécutive provenant du même nœud. Si le délai du temporisateur expire, l'objet Keeper maître doit automatiquement libérer la ressource réseau.

Un objet Keeper de sauvegarde doit répondre aux demandes de service de ressource réseau comme suit. En cas de réception d'une demande de service *Obtain_Network_Resource*, un temporisateur de 60 s doit commencer, puis être déclenché lors de la réception d'une demande de service *Hold_Network_Resource*. Le temporisateur doit également être démarré lorsqu'un objet Keeper est activé sur un réseau existant et qu'il constate que la ressource réseau est en cours d'obtention. Le temporisateur doit être arrêté lorsqu'un service *Release_Network_Resource* est reçu ou que le temporisateur expire.

Lorsqu'un objet Keeper de sauvegarde devient un objet Keeper maître, il doit utiliser les données provenant du dernier Lpacket TUI reçu afin de récupérer la ressource réseau de l'ancien objet Keeper maître.

Eu égard aux objets Keeper maître et de sauvegarde, le temporisateur de 60 s doit permettre de déterminer si le nœud qui demande la ressource réseau n'est pas tombé en panne ou a été prématurément retiré du réseau. Lorsque le temporisateur de 60 s d'un objet Keeper expire, l'objet Keeper doit

- a) annuler toutes les modifications d'attribut en cours;
- b) réinitialiser le bit d'état TUI de variation nette en cours;
- c) quitter l'état de fonctionnement de variation nette;
- d) reprendre le fonctionnement normal.

Étant donné que la ressource réseau peut être conservée pendant une période assez longue si l'utilisateur procède à des modifications importantes, elle doit être clairement transférée vers le nouvel objet Keeper maître en cas de modification de l'objet Keeper maître. Ce transfert doit être invisible au nœud pour lequel l'objet Keeper maître gère la ressource réseau. Étant donné que toutes les informations relatives à la ressource réseau sont contenues dans le Lpacket TUI que l'objet Keeper maître diffuse, et que les objets Keeper de sauvegarde reçoivent le TUI diffusé et l'utilisent pour déterminer s'il convient qu'ils jouent le rôle d'objet Keeper maître et à quel moment ils doivent le faire, les objets Keeper de sauvegarde doivent détenir les informations nécessaires pour transmettre correctement la ressource.

L'objet Keeper de sauvegarde et l'objet Keeper maître défaillants doivent traiter les demandes de ressource réseau comme s'ils ne l'étaient pas.

Si la ressource réseau n'est pas traitée, les champs connexes des Lpackets TUI de réseau doivent être réinitialisés:

- NR_VENDOR_ID;
- NR_SERIAL_NUMBER;
- NR_CLASS;
- NR_INSTANCE.

7.3.6.3 Service Obtain_Network_Resource

La demande de service *Obtain_Network_Resource* doit permettre à l'objet Keeper maître de gérer la ressource réseau pendant 60 s pour lui-même ou un autre nœud, s'il n'est pas déjà géré. La période de gestion de la ressource réseau peut être étendue de manière indéfinie par l'utilisation de la demande de service *Hold_Network_Resource*.

Si la ressource réseau est déjà gérée pour un autre nœud, un état d'erreur doit être renvoyé.

Le service Obtain_Network_Resource doit renvoyer les codes d'erreur (voir Tableau 29) comme indiqué dans la colonne ONR pour ce service.

7.3.6.4 Service Hold_Network_Resource

La demande de service Hold_Network_Resource doit permettre à l'objet Keeper maître de continuer à gérer la ressource réseau pendant les 60 s qui suivent si le nœud qui le demande est celui pour lequel l'objet Keeper maître est en train de gérer la ressource réseau.

Si la ressource réseau est gérée pour un autre nœud ou qu'elle n'est pas gérée, un état d'erreur doit être renvoyé.

Le service Hold_Network_Resource doit renvoyer les codes d'erreur (voir Tableau 29) comme indiqué dans la colonne HNR pour ce service.

7.3.6.5 Service Release_Network_Resource

La demande de service Release_Network_Resource doit permettre à l'objet Keeper maître d'arrêter la ressource réseau si le nœud qui demande la libération est celui pour lequel l'objet Keeper maître gère la ressource réseau. Si une modification de réseau est en cours et que la ressource réseau est libérée, la modification est annulée.

Si la ressource réseau est gérée pour un autre nœud ou qu'elle n'est pas gérée, un état d'erreur doit être renvoyé.

Le service Release_Network_Resource doit renvoyer les codes d'erreur (voir Tableau 29) comme indiqué dans la colonne RNR pour ce service.

7.3.6.6 Service Change_Start

La demande de service Change_Start doit permettre à l'objet Keeper:

- a) de copier l'exemplaire en cours des attributs de la mémoire non volatile dans l'exemplaire en attente de la mémoire RAM;
- b) d'annuler le nombre de demandes de service Set_Attribute (y compris individuel, liste et fragment) reçues par l'objet ControlNet;
- c) de placer l'objet Keeper à l'état de fonctionnement net_change approprié.

Cette demande de service doit uniquement être traitée si la ressource réseau est gérée par l'objet Keeper maître comme indiqué par le bit net_resource dans le Lpacket TUI diffusé.

Le service Change_Start doit renvoyer les codes d'erreur (voir le Tableau 29) comme indiqué dans la colonne CS pour ce service.

7.3.6.7 Service Change_Complete

La demande de service Change_Complete doit permettre à l'objet Keeper:

- a) de copier l'exemplaire en attente des attributs de la mémoire RAM vers l'exemplaire en cours des attributs de la mémoire non volatile;
- b) de procéder à une modification de réseau synchronisée;
- c) de vérifier que tool_keeper_revision de l'attribut 105 est $\leq$ révision de l'objet Keeper (si ce n'est pas le cas, marquer le tableau d'attribut comme étant non valide);
- d) de quitter l'état de fonctionnement de modification de réseau dans lequel il se trouve;
- e) de reprendre son fonctionnement normal.

Si l'objet Keeper ne se trouve pas dans l'un des états de fonctionnement de variation nette, la demande de service Change_Complete doit être ignorée et une réponse d'erreur d'état incorrect générée.

Si le nombre de demandes de service Set_Attribute (notamment Set_Attribute_Single, Set_Attribute_List et Set_Attribute_Fragment) reçues par l'objet Keeper depuis la demande de service Change_Start ne correspond pas à celui du paramètre Change_Complete, un Change_Abort doit être exécuté et une réponse d'erreur générée.

La réponse à ce service doit être renvoyée après la diffusion d'au moins un TUI, le bit net_change_in_progress étant annulé. Une modification synchronisée doit avoir lieu avant la transmission TUI avec le bit remis à zéro.

Le service Change_Complete doit renvoyer les codes d'erreur (voir le Tableau 29) comme indiqué dans la colonne CC pour ce service.

7.3.6.8 Service Change_Abort

Le service Change_Abort doit permettre à l'objet Keeper:

- a) d'écarter les modifications apportées aux données d'attribut en attente;
- b) de libérer la ressource réseau;
- c) de reprendre son fonctionnement normal.

La réponse à ce service doit être renvoyée après la diffusion d'au moins un TUI, avec le bit net_change_in_progress, et le bit holding_network_resource étant annulé.

Le service Change_Abort doit renvoyer les codes d'erreur (voir le Tableau 29) comme indiqué dans la colonne CA pour ce service.

7.3.6.9 Service Get_Signature

Le service Get_Signature doit permettre à l'objet Keeper maître de consulter les valeurs de mot de passe CO de l'auteur d'une connexion dans l'attribut CO_data. Le chemin vers le dispositif doit être l'attribut de la demande. Ce chemin doit faire l'objet d'une analyse syntaxique et utilisé pour rechercher la structure en arborescence des données CO_data, une branche à la fois, afin de détecter la valeur du mot de passe CO approprié, qui est renvoyée en tant que paramètre dans la réponse.

Le chemin doit faire l'objet d'une analyse syntaxique une branche à la fois, étant donné que les entrées du chemin de l'attribut CO_data spécifient uniquement la partie du chemin entre deux branches, et pas l'ensemble du chemin vers cette branche.

Le service Change_Signature doit renvoyer les codes d'erreur (voir le Tableau 29) comme indiqué dans la colonne GS pour ce service.

7.3.6.10 Service/réponse Get_Attribute_Fragment

Le service Get_Attribute_Fragment doit collecter et renvoyer une partie des données d'attribut CO_data gérées par l'objet Keeper. Toutes les parties de cet attribut peuvent être lues. L'attribut CO_data doit être le seul attribut Keeper qui répond aux services de fragment. L'accès aux autres attributs doit générer une erreur à renvoyer.

NOTE Lors de l'obtention de données d'attribut, il convient de renvoyer les données d'attribut en cours issues de la mémoire non volatile, et pas l'exemplaire en attente de l'attribut.

Le service Get_Attribute_Fragment doit renvoyer les codes d'erreur (voir le Tableau 29) comme indiqué dans la colonne GAF pour ce service.

7.3.6.11 Service Set_Attribute_Fragment

Le service Set_Attribute_Fragment doit être utilisé pour définir une partie de l'attribut CO_data géré par l'objet Keeper. Seul l'attribut CO_data doit être accessible par ce service. L'accès aux autres attributs doit générer une erreur à renvoyer.

Le service Set_Attribute_Fragment doit être traité uniquement lorsque l'objet Keeper se trouve dans l'un des états de fonctionnement de variation nette. L'objet Keeper doit renvoyer une erreur de mode non valide s'il ne se trouve pas dans l'un des états de fonctionnement de variation nette lors de la réception d'une demande de service Set_Attribute_Fragment.

La définition des données d'attribut doit définir l'exemplaire en attente de l'attribut, pas l'exemplaire en cours de la mémoire non volatile. Les attributs en attente doivent être copiés dans les attributs en cours de la mémoire non volatile dès la réception d'une demande de service Change_Complete spécifique à la classe.

L'objet Keeper ne doit pas vérifier la valeur d'attribut pendant une opération de définition.

Le service Set_Attribute_Fragment doit renvoyer les codes d'erreur (voir le Tableau 29) comme indiqué dans la colonne SAF pour ce service.

7.3.6.12 Identifiant unique de table (diffusion), balise fixe 0x84

Si l'objet Keeper est à l'état de fonctionnement master_verify, faulted_master_verify, master, faulted_master, net_change_master ou net_change_faulted_master, il doit tenter de transmettre un TUI (Identifiant unique de table) Lpacket particulier en tant que données planifiées une fois toutes les 4 NUT (c'est-à-dire les numéros NUT 0, 4, 8, 12, 16 ...). Tous les dispositifs Keeper doivent réserver 26 octets de durée de transmission de liaison planifiée une fois toutes les 4 NUT pour transmettre ce Lpacket (en cas d'envoi non planifié, il doit s'agir des informations non planifiées de priorité QoS la plus élevée).

Lors de la génération du TUI Lpacket transmis, tous les champs réservés doivent utiliser les valeurs spécifiées dans les champs réservés correspondants de l'attribut 0x0101. L'outil de programmation doit affecter une valeur nulle à tous les champs réservés de l'attribut 0x0101.

Le format des données de liaison TUI Lpacket doit être tel que défini dans le Tableau 30.

Tableau 30 – Format de virement du TUI Lpacket

Nom	Type de données	Description du paramètre	Sémantique des valeurs
Size	USINT	Taille du Lpacket, en mots.	0x0D
Contrôle	USINT	Octet de contrôle du Lpacket de couche de liaison.	0x01 (Lpacket à balise fixe)
Fixed_Tag	USINT	Valeur de balise fixe.	0x84 (TUI Lpacket)
Destination_Mac_Id	USINT	Destinataire du Lpacket.	0xFF (Diffusion)
Unique_Id	UDINT	CRC des attributs en cours de Keeper.	Octet de poids faible en premier.
Status_Flag	UINT	Valeurs de balise TUI.	Voir l'attribut TUI pour plus de détails.
Keeper_Mac_Id	USINT	MAC ID du Keeper diffusant le TUI.	Voir l'attribut TUI pour plus de détails.
Réservé	USINT	Réservé pour l'alignement des données	
Net_Resource_Vendor_Id	UINT	ID fournisseur de l'objet détenant le Net Resource pour une utilisation exclusive	
Net_Resource_Serial_Number	UDINT	Numéro de série de l'objet détenant le Net Resource pour une utilisation exclusive	
Net_Resource_Class	UDINT	Numéro de classe de l'objet détenant le Net Resource pour une utilisation exclusive	
Net_Resource_Instance	UDINT	Numéro d'instance de l'objet détenant le Net Resource pour une utilisation exclusive	

7.3.7 Codes d'erreur de service

Le Tableau 31 répertorie les codes d'erreur possibles et les conditions les plus probables dans lesquelles le code est renvoyé.

Tableau 31 – Codes d'erreur du service

Code d'erreur	Signification	Condition de renvoi
0x02	resource_unavailable_err	La ressource réseau demandée n'est pas disponible
0x03	invalid_parameter_err	Un paramètre non valide a été fourni au service
0x04	path_segment_err	A chaque spécification d'un chemin supérieur au numéro d'attribut
0x05	path_dest_unknown_err	L'instance d'objet n'existe pas
0x08	unimplemented_service_err	A chaque fois que le service n'est pas pris en charge pour un attribut
0x09	invalid_attr_err	A chaque fois qu'un attribut spécifié n'est pas pris en charge dans cet objet
0x0C	wrong_object_state_err	L'état de l'objet ne permet pas d'exécuter le service
0x0D	already_exists_err	Le service a déjà été obtenu par le nœud demandeur
0x0E	not_settable_err	L'attribut ne peut pas être défini
0x0F	no_permission_err	Le nœud demandant ce service ne détient pas la ressource
0x11	reply_too_large_err	Espace insuffisant dans la mémoire tampon de réponse
0x13	not_enough_data_err	A chaque fois que la demande ne contient pas assez de données
0x14	undefined_attr_err	Tentatives d'accès à un attribut non défini
0x15	too_much_data_err	Plus de données que prévu ont été rencontrées pour cette demande de service
0x16	nonexistent_object_err	Objet Keeper non disponible
0x26	invalid_path_size_err	A chaque fois qu'un segment de chemin non valide est spécifié pour ce service

7.3.8 Comportement

L'indication numérique de l'état de l'objet Keeper doit être telle que définie dans le Tableau 32.

Tableau 32 – États de fonctionnement de l'objet Keeper

État	Description
0,1,2 – Mise sous tension	L'objet Keeper attend la mise en ligne du nœud ou détermine s'il s'agit d'un objet Keeper légitime pour cette liaison
3 – Sauvegarde	L'objet Keeper a déterminé qu'il s'agit d'un objet Keeper légitime pour cette liaison, mais a déterminé qu'il s'agit d'un objet Keeper de sauvegarde ou n'a pas encore déterminé s'il s'agit de l'objet Keeper maître pour cette liaison
4 – Vérification du maître	L'objet Keeper a déterminé qu'il s'agit d'un objet Keeper légitime pour cette liaison, mais n'a pas entendu d'autres objets Keeper actifs sur la liaison, a reçu un TUI correct provenant d'un objet Keeper correspondant avec un numéro de nœud supérieur ou a entendu un TUI provenant d'un objet Keeper défaillant. Il commence à diffuser le TUI Lpacket, mais ne traite pas les modifications de paramètre synchronisées ni ne répond aux demandes de l'objet Keeper maître
5 – Maître	L'objet Keeper a déterminé qu'il s'agit d'un objet Keeper légitime et qu'il s'agit de l'objet Keeper maître pour cette liaison. Il diffuse le TUI Lpacket, traite les modifications de liaison synchronisées, le cas échéant, pour modifier les paramètres de liaison et doit répondre aux demandes de l'objet Keeper maître
6 – Échec de la sauvegarde	L'objet Keeper a déterminé qu'il ne s'agit pas d'un objet Keeper légitime pour cette liaison, et qu'il n'est pas l'objet Keeper si la liaison ne contient que des objets Keeper défaillants
7 – Échec de la vérification du maître	L'objet Keeper est défaillant, mais n'a pas entendu un autre objet Keeper actif sur le réseau pendant 12 NUT. Il commence à diffuser le TUI Lpacket, mais ne traite pas les modifications de liaison synchronisées ni ne répond aux demandes de l'objet Keeper maître
8 – Échec du maître	L'objet Keeper est défaillant et a supposé les fonctions de l'objet Keeper maître. Le réseau ne contient aucun objet Keeper non défaillant. Il diffuse le TUI Lpacket, traite les modifications de liaison synchronisées, le cas échéant, pour modifier les paramètres de réseau et doit répondre aux demandes de l'objet Keeper maître

État	Description
9,10,11,12 – Variation nette	Les attributs de l'objet Keeper sont mis à jour. Il existe un sous-état <i>Variation nette</i> pour les états <i>sauvegarde</i> , <i>maître</i> , <i>défaut</i> et <i>échec du maître</i> préalablement indiqués. Les sous-états séparés sont nécessaires pour permettre à l'objet Keeper de pouvoir quitter l'état <i>Variation nette</i> correctement lorsque des mises à jour se terminent normalement et qu'elles sont annulées. Les attributs en attente peuvent uniquement être définis lorsque l'objet Keeper se trouve à l'état <i>Variation nette</i> . Les nœuds du <i>maître de variation nette</i> et du <i>maître défaillant de variation nette</i> doivent répondre aux demandes de l'objet Keeper maître. Les nœuds des états <i>sauvegarde de variation nette</i> ou <i>sauvegarde défaillante de variation nette</i> ne doivent pas répondre aux demandes de l'objet Keeper maître.

7.3.9 Notes diverses

Les objets Keeper doivent gérer les paramètres des nœuds de la même liaison uniquement.

Chaque objet Keeper doit être en mesure de distribuer les paramètres de liaison à tous les autres nœuds de cette liaison. L'objet Keeper maître doit être l'objet Keeper dont le MAC ID est le plus bas parmi tous les objets Keepers légitimes de la liaison. Un objet Keeper maître défaillant sur une liaison peut ou peut ne pas être le MAC ID le plus bas.

L'objet Keeper doit être chargé de la configuration des dispositifs qui apparaissent dans ces attributs. L'objet Keeper maître doit être le seul objet Keeper qui réponde aux demandes de service par l'intermédiaire de la balise fixe 0x88 et diffusées à tous les objets Keeper d'un réseau.

Tous les objets Keeper légitimes d'une liaison doivent détenir des exemplaires identiques des attributs de réseau pour tous les dispositifs de la liaison. Les objets Keeper qui n'acceptent pas l'objet Keeper maître doivent rester à l'état d'échec de la sauvegarde tant qu'ils n'ont pas changé (accepter signifie que leurs paramètres TUI *unique_id* sont identiques).

Un seul terminal de programmation peut modifier les attributs Keeper simultanément. Le terminal de programmation doit acquérir la ressource réseau avant de modifier l'objet Keeper, et doit toujours être en possession de la ressource réseau pendant toute la procédure de modification. Si un terminal de programmation modifie un objet Keeper, il doit mettre à jour tous les autres objets Keeper de la liaison, et diffuser les demandes de service Change_Start, Set_Attribute et Change_Stop.

L'objet Keeper doit vérifier la révision de l'outil Keeper dans l'attribut 0x0104 pour un tableau d'attributs valide. Si l'objet Keeper n'est pas en mesure de comprendre cette révision, il doit supprimer le bit 5 des transmissions TUI, de manière à permettre à d'autres objets Keeper ayant cette capacité de compréhension d'assumer les responsabilités d'objet Keeper maître. Les révisions 0 et 1 de l'objet Keeper doivent être identiques à la révision 2.

La valeur de révision d'informations de connexion est indépendante de la révision de l'objet Keeper et doit être utilisée par les outils de programmation.

Pour les besoins de diagnostic du réseau, l'objet Keeper peut diffuser un Lpacket à balise fixe de débogage (0x90) à tous les nœuds avec trois mots de données. Le premier mot identifie que l'objet Keeper envoie le diagnostic. Le deuxième mot est l'état en cours de l'objet Keeper, le troisième mot étant l'état suivant de l'objet Keeper.

7.3.10 Séquence d'activation de l'objet Keeper

7.3.10.1 Généralités

La séquence d'activation de l'objet Keeper doit lui permettre de commencer des opérations dans un large éventail de conditions, en passant d'un état d'activation de réseau configuré normal à un état dans lequel de nombreux dispositifs de réseau sont nouveaux et non configurés.

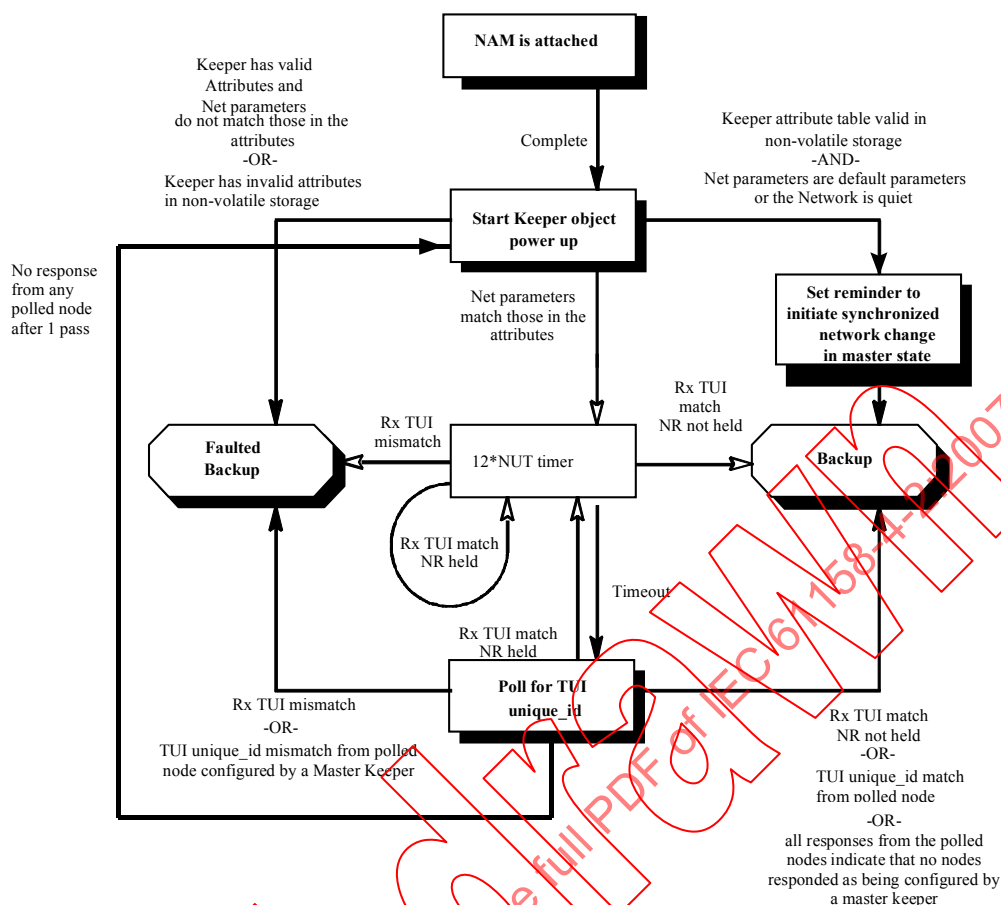
La séquence d'activation de l'objet Keeper ne doit pas commencer tant que le moniteur d'accès au réseau n'a pas mis le nœud Keeper en ligne. L'activation de l'objet Keeper doit déterminer si ce dispositif Keeper particulier est un objet Keeper légitime pour la liaison locale et placer l'objet Keeper à l'état de fonctionnement de sauvegarde ou d'échec de la sauvegarde.

S'il s'agit d'un objet Keeper légitime pour la liaison, l'objet Keeper doit déterminer s'il s'agit d'un objet Keeper maître ou d'un objet Keeper de sauvegarde. S'il ne s'agit pas d'un objet Keeper légitime, l'objet Keeper passe à l'état d'anomalie. Un objet Keeper non configuré doit passer à l'état d'anomalie à l'activation.

Le temporisateur 12*NUT doit être de nouveau déclenché à chaque réception d'un TUI Lpacket indiquant que la ressource réseau est gérée. Il s'agit d'indiquer que les attributs Keeper sont en cours de mise à jour.

La séquence d'activation de l'objet Keeper est illustrée dans la Figure 18.

Withdrawing
IECNORM.COM: Click to view the full PDF of IEC 61158-4-2:2007



Légende

Anglais	Français
NAM is attached	NAM est associé
Keeper has valid ... storage	L'objet Keeper a des attributs valides et les paramètres Net ne correspondent pas à ceux des attributs –OU- l'objet Keeper a des attributs invalides dans la mémoire non volatile
Complete	Complet
Keeper attribute ... is quiet	Table d'attributs de l'objet Keeper valide dans la mémoire non volatile –ET- les paramètres Net sont des paramètres défaillants ou le réseau est calme
Start keeper object power up	Démarrage de l'activation de l'objet Keeper
No response ... 1 pass	Aucune réponse de tout nœud interrogé après 1 passage
Net parameters ... attributes	Les paramètres Net correspondent à ceux des attributs
Set reminder ... state	Définir le reste pour initier les modifications de réseau synchronisées à l'état maître
Faulted backup	Echec de sauvegarde
Rx TUI mismatch	Défaut d'adaptation Rx TUI
12*NUT timer	Temporisateur 12 NUT
Rx TUI match NR not held	Correspondance Rx TUI NR non détenu
Backup	Sauvegarde
Rx TUI match NR held	Correspondance Rx TUI NR détenu
Timeout	Temporisation
-OR-	-OU-
TUI unique_id mismatch ... keeper	Non correspondance du TUI unique_id avec le nœud interrogé configuré par un objet Keeper maître

Anglais	Français
Poll for	Interrogation du TUI unique_id
TUI unique_id match from polled node	Correspondance du TUI unique_id avec le nœud interrogé
All responses ... keeper	Toutes les réponses du nœud interrogé indiquent qu'aucun nœud n'a répondu en ayant été configuré par l'objet Keeper maître

Figure 18 – Diagramme d'états d'activation de l'objet Keeper

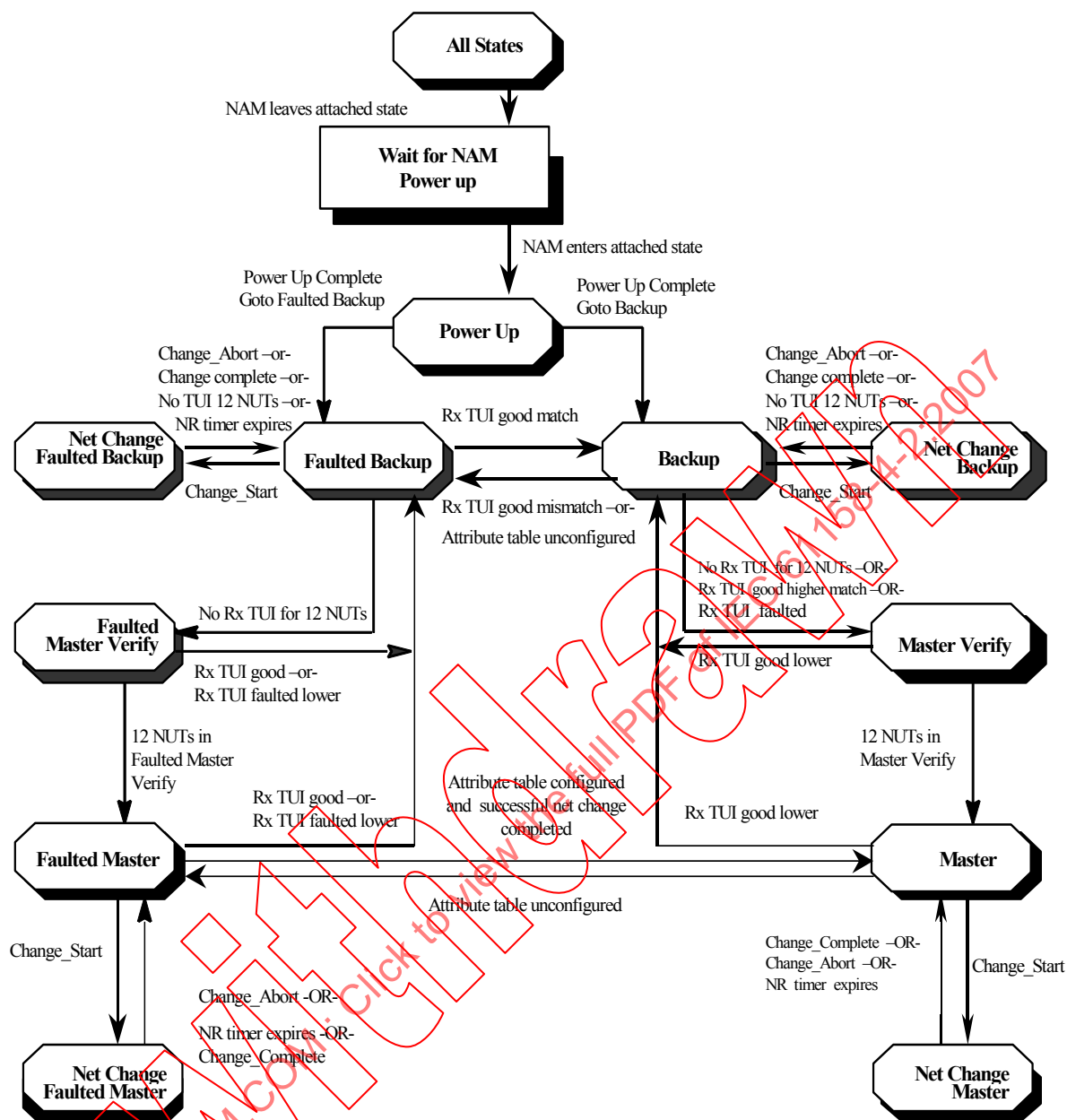
7.3.10.2 Interrogation de TUI unique_id

L'état d'interrogation de TUI unique_id de la Figure 18 doit impliquer l'opération suivante, réalisée par l'objet Keeper:

- Émettre une demande UCMM à l'objet ControlNet du nœud adressé pour lire les paramètres de configuration en cours de l'objet. Si la demande n'aboutit pas, passer au MAC ID suivant tant que UMAX n'est pas atteint. Plusieurs demandes simultanées sont admises pour réduire la durée d'interrogation.
- Chaque MAC ID de la plage 1 à UMAX (à l'exclusion du MAC ID du nœud Keeper) peut être interrogé individuellement ou simultanément avec une demande Get_Attribute_Single sur l'attribut current_link_config de l'objet ControlNet du nœud. L'attribut contient un exemplaire de l'attribut TUI en cours pour le nœud. L'accès simultané à plusieurs nœuds permet d'accélérer le processus d'interrogation.
- Si aucune réponse n'est obtenue d'un nœud interrogé et que UMAX a été atteint, l'interrogation est annulée et l'objet Keeper doit passer à l'état d'activation de l'objet Keeper, étant donné qu'au moins un autre nœud d'un réseau non défaillant est présent mais incapable de répondre. Cela permet de répéter le processus d'interrogation d'activation tant qu'un nœud n'est pas en mesure de répondre.
- Si une réponse a été reçue d'un nœud interrogé et qu'une valeur a été attribuée au bit d'état de l'objet Keeper dans le TUI status_flag renvoyé, le nœud interrogé a été configuré par un objet Keeper à l'état de fonctionnement Maître. Comparer la valeur du TUI unique_id renvoyée par le nœud interrogé à celle de la table d'attributs de cet objet Keeper. Si elles correspondent, l'interrogation est annulée et l'objet Keeper doit passer à l'état de fonctionnement de sauvegarde. Si elles ne correspondent pas, l'interrogation est annulée et l'objet Keeper doit passer à l'état de fonctionnement d'échec de la sauvegarde.
- Si une réponse a été reçue d'un nœud interrogé et qu'une valeur n'a pas été attribuée au bit d'état de l'objet Keeper dans le TUI status_flag renvoyé, le nœud interrogé n'a pas été configuré par un objet Keeper à l'état de fonctionnement Maître. Dans ce cas, les informations sont écartées et l'interrogation se poursuit. Si tous les nœuds répondent de cette manière, indiquant que l'un d'eux a été configuré par un objet Keeper à l'état de fonctionnement maître, l'interrogation se termine et l'objet Keeper doit passer à l'état de fonctionnement de sauvegarde, puisque dans ce cas, tous les nœuds sont nouveaux ou ont perdu leur configuration stockée.

7.3.10.3 États de fonctionnement

Les états de fonctionnement de l'objet Keeper doivent être définis conformément à la Figure 19, au Tableau 33 et aux sections ci-dessous.



Légende

Anglais	Français
All states	Tous les états
NAM leaves attached state	Le NAM quitte l'état associé
Wait for NAM power up	Attente de l'activation du NAM
NAM enters attached state	Le NAM passe à l'état associé
Power up complete	Activation réalisée
Goto faulted backup	aller à l'état échec de sauvegarde
Goto backup	Aller à l'état de sauvegarde
-or-	-ou-
No TUI 12 NUTs	Pas de TUI de 12 NUT
NR timer expires	Temporisateur NR écoulé
Rx TUI good match	Bonne correspondance Rx TUI
Net change	Variation nette

Anglais	Français
faulted backup	échec de sauvegarde
Backup	Sauvegarde
Rx TUI good mismatch	Non correspondance Rx TUI correcte
Attribute table unconfigured	Table d'attributs non configurée
No Rx TUI for 12 NUTs	Pas de Rx TUI pendant 12 NUT
Rx TUI good higher match	Bonne correspondance supérieure Rx TUI
Rx TUI faulted	Echec Rx TUI
Faulted	Echec
Master verify	Vérification maître
Rx TUI good	Rx TUI correcte
Rx TUI faulted lower	Echec Rx TUI inférieure
Rx TUI good lower	Rx TUI inférieure correcte
12 NUTs in faulted master verify	12 NUT au cours de l'échec de vérification maître
12 NUTs in master verify	12 NUT au cours de la vérification maître
Rx TUI good	Rx TUI correcte
Faulted master	Echec maître
Master	Maître
Attribute table configured ... completed	Table d'attributs configurée et réussite de la réalisation de la variation nette
Net change faulted master	Variation nette - échec maître
Net change master	Variation nette - maître

Figure 19 – Diagramme d'états de fonctionnement de l'objet Keeper

Tableau 33 – Matrice d'événement d'état de l'objet Keeper

Événement	État											
	Activati on	Sauveg arde	Vérificat ion du maître	Maître	Échec de la sauvegard e	Échec de la vérification du maître	Échec du maître	Variation nette (sauvega rde)	Variation nette (Ma ître)	Variation nette (Éc hec de la sauvegar de)	Variation nette (Échec du maître)	
Rx TUI provenant d'un nœud correct						Aller à Échec de la sauvegard e	Aller à Échec de la sauvegard e					
Rx TUI provenant d'un nœud inférieur correct			Aller à Sauvegarde									
Rx TUI provenant d'un nœud supérieur correct (correspondance)		Aller à Vérificat ion du maître										
Rx TUI provenant d'un nœud correct (non correspondance)		Aller à Échec de la sauvega rde										
Rx TUI provenant d'un nœud correct (correspondance)					Aller à Sauvegard e							
Rx TUI provenant d'un nœud défaillant		Aller à Vérificat ion du maître										
Rx TUI provenant d'un nœud inférieur défaillant						Aller à Échec de la sauvegard e						
Pas de TUI Rx pendant 12*NUT		Aller à Vérificat ion du maître			Aller à Échec de la vérification du maître			Annuler modificati on Aller à Sauvegar de		Annuler modificati on Aller à Échec de la sauvegar de		
12 Nut écoulées			Aller à Maître			Aller à Échec du maître						
Change_Start		Aller à Variatio n nette (Sauveg arde)		Aller à Variatio n nette (M aître)	Aller à Variation nette (Échec de la sauvegard e)		Aller à Variation nette (Échec du maître)					
Change_Complete								Aller à Sauvegar de	Aller à Maître	Aller à Sauvegar de	Aller à Vérificati on du maître	
Change_Abort								Annuler modificati on Aller à Sauvegar de	Annuler modificati on Aller à Maître	Annuler modificati on Aller à Échec de la sauvegar de	Annuler modificati on. Aller à Échec du maître	

Événement	État										
Le temporisateur de ressource réseau 60 s a expiré								Annuler modification Aller à Sauvegarde	Annuler modification Aller à Maître	Annuler modification Aller à Échec de la sauvegarde	Annuler modification Aller à Échec du maître
Le moniteur d'accès au réseau a quitté l'état associé	Attendre l'activation du moniteur d'accès au réseau										
Activation terminée pour Sauvegarde	Aller à Sauvegarde										
Activation terminée pour Échec de la sauvegarde	Aller à Échec de la sauvegarde										
NOTE Les cellules vides indiquent que cet événement ne peut pas se produire ou que, s'il se produit, aucune action ne doit être entreprise par l'objet Keeper.											

7.3.10.4 Ressource réseau

L'accès aux objets Keeper de modification doit être contrôlé à l'aide de la ressource réseau. Un seul dispositif peut acquérir la ressource réseau pour un accès exclusif à la fois.

La ressource réseau doit être gérée par l'objet Keeper maître pour le nœud qui procède aux mises à jour de l'attribut. Si l'objet Keeper maître n'écoute pas le nœud qui procède à la mise à jour au moins une fois toutes les 60 s, il doit automatiquement libérer la ressource réseau, annuler la modification en cours et reprendre un fonctionnement normal.

7.3.10.5 Keeper défaillant

Un objet Keeper dont l'attribut *TUI unique_id* ne correspond pas à celui de la diffusion TUI provenant d'un objet Keeper doit passer à l'état d'anomalie, sauf si une opération de variation nette est en cours.

7.3.10.6 Le MAC ID choisit l'objet Keeper maître

L'objet Keeper doit utiliser le MAC ID du nœud Keeper inclus dans le message TUI pour déterminer l'objet Keeper d'une liaison qui est le maître.

7.3.10.7 Vérification du maître

L'objet Keeper doit attendre 12*NUT après le début de la première diffusion TUI avant de supposer l'état de l'objet Keeper maître. Cette durée doit être suffisante pour permettre à un objet Keeper doté d'un MAC ID inférieur de devenir le maître. Les autres objets Keeper peuvent devenir des objets Keeper de sauvegarde.

7.3.10.8 Suspect

Si le nœud contenant l'objet Keeper devient suspect pendant une séquence de modification de réseau synchronisée, le bit de variation nette en cours doit être supprimé, et le contrôle doit être renvoyé au début du traitement de la modification de réseau synchronisée immédiatement après que le moniteur d'accès au réseau a remis le nœud en ligne. La séquence normale d'activation de l'objet Keeper doit être ignorée.

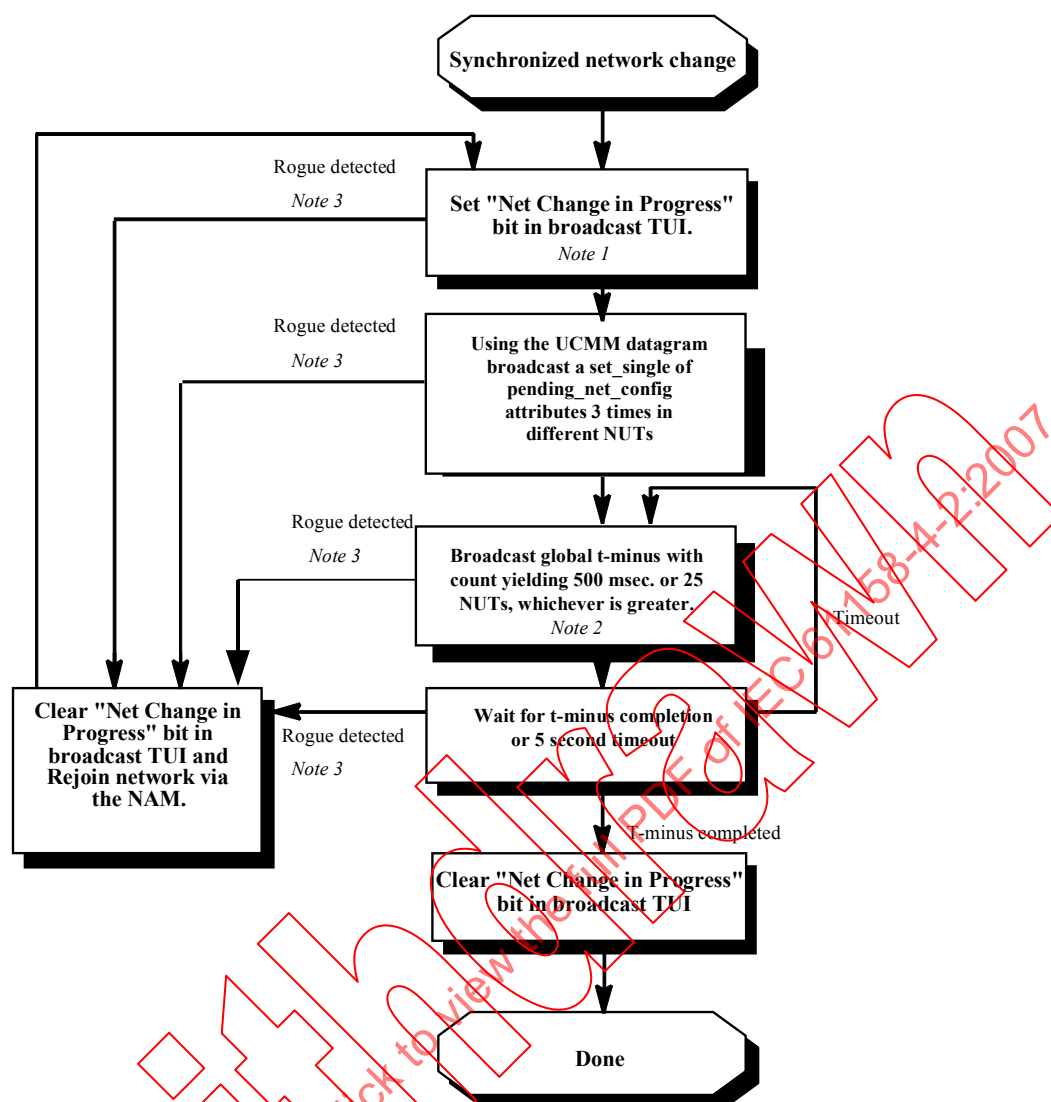
7.3.10.9 Algorithme de traitement de la modification de réseau synchronisée

Bien que la liaison utilise le modérateur pour distribuer les paramètres de fonctionnement, il revient à l'objet Keeper de modifier les paramètres de liaison.

Toutes les modifications de paramètre de liaison doivent être réalisées à l'aide d'un algorithme de modification de réseau synchronisé. Cela permet à tous les nœuds du réseau de modifier simultanément leurs paramètres de liaison en cours sans interrompre le fonctionnement du réseau.

L'algorithme de modification de réseau synchronisée doit être tel que défini dans la Figure 20.

Withdrawing
IECNORM.COM: Click to view the full PDF of IEC 61158-4-2:2007



Légende

Anglais	Français
Synchronized network change	Modification de réseau synchronisée
Rogue detected	Événement suspect détecté
Set ... TUI	Définir le bit modification de réseau en cours dans le TUI de diffusion
Using the ... NUTs	A l'aide du datagramme UCM, diffuser un set_single des attributs pending_net_config 3 fois pendant différentes NUT
Broadcast ... greater	Diffuser t-minus global à un rythme de 500 msec. ou 25 NUT, selon la plus grande valeur
Timeout	Temporisation
Clear "net change in progress" bit in broadcast TUI	Effacer le bit " modification de réseau en cours" dans le TUI de diffusion
And rejoin network via the NAM	Et joindre le réseau via le NAM
Wait for ... timeout	Attendre la réalisation complète de t-minus ou une temporisation de 5 secondes
t-minus completed	t-minus écoulé
Done	Fin réalisation

NOTE 1 La diffusion d'un TUI avec le bit de modification de réseau en cours oblige les nœuds qui viennent d'être activés à suspendre la mise en ligne tant que la modification de réseau n'est pas terminée, et que le bit de

modification de réseau en cours de la diffusion TUI n'est pas supprimé. Cela permet de limiter le risque d'événement suspect.

NOTE 2 Le délai t_{Minus} assure l'immunité au bruit et permet à tous les nœuds d'entendre au moins un modérateur avec le nouveau comptage t_{Minus} . Il offre également aux nœuds une durée de traitement raisonnable des paquets Set_Single avec de nouveaux attributs de réseau. Le délai est déterminé en prenant la valeur maximale 25 ou en divisant 500 ms par la durée NUT en cours (en ms).

NOTE 3 La condition suspecte se produit uniquement si le nœud modérateur et au moins un autre nœud n'ont pas reçu l'un des trois paquets de demande de service de l'attribut de réseau de définition de diffusion. Cela se produit uniquement en cas d'événement bruyant sérieux (durant au moins 3 NUT).

Figure 20 – Traitement de la modification de réseau synchronisée

7.4 Objet de planification

7.4.1 Présentation

L'objet de planification doit exister dans les dispositifs à l'origine de la connexion (CO). L'objet de planification doit être utilisé par un logiciel de planification de liaison (LSS) pour

- lire les informations de connexion planifiée;
- écrire les informations de planification;
- offrir un mot de passe (clé logicielle) à l'auteur de la connexion, qui doit être utilisé pour authentifier l'accès du dispositif CO à une liaison particulière.

Une session LSS avec l'objet de planification doit permettre de planifier une liaison particulière. Si un dispositif CO détient des connexions sur plusieurs liaisons, plusieurs sessions LSS doivent être nécessaires pour planifier la totalité des connexions. Les résultats de la session de planification doivent être sauvegardés par le dispositif CO (y compris le mot de passe CO calculé par le LSS).

Étant donné que le LSS peut être intégré au logiciel de programmation d'un dispositif CO, l'objet de planification doit fournir les commandes de prise en charge d'un logiciel de programmation intégré ou séparé. De manière spécifique, lorsque le LSS écrit des données de planification dans l'objet de planification, il peut le faire sous condition ou forcer l'écriture. Les données de planification doivent être écrites sous condition si le logiciel de programmation et le LSS n'ont pas été intégrés, et l'objet de planification doit uniquement utiliser les données de planification fournies si un autre logiciel de programmation n'a pas modifié les informations de connexion pendant la session LSS. Une écriture forcée peut être utilisée avec un LSS/logiciel de programmation intégré si l'outil intégré peut garantir qu'aucun autre logiciel de programmation n'a modifié les informations de connexion depuis la dernière lecture pendant la session de planification en cours.

Toutes les communications adressées à l'objet de planification doivent utiliser le service fixe UCMM (Unconnected Message Manager) ou un service de connexion à balise générique actif vers l'objet Routeur de message.

7.4.2 Attributs de classe

L'objet de planification doit prendre en charge les attributs de classe spécifiés dans le Tableau 34. Le service Get_Attribute_All doit renvoyer les attributs de niveau de classe dans l'ordre numérique croissant.

Tableau 34 – Attributs de classe de l'objet de planification

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut
0x01	Requise	Get	Revision	UINT	Première révision, valeur = 1
0x02	Requise	Get	MaxInstances	UDINT	Nombre maximal d'objets de planification pris en charge
0x03	Requise	Get	NumInstances	UDINT	Nombre d'objets de planification instanciés

7.4.3 Attributs d'instance

L'objet de planification doit prendre en charge les attributs d'instance spécifiés dans le Tableau 35. Le service Get_Attribute_All doit renvoyer les attributs de niveau d'attribut dans l'ordre numérique croissant.

Tableau 35 – Attributs d'instance de l'objet de planification

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut
0x01	Requise	Get/Set	Route	STRUCT de taille variable	Route entre le CO et la liaison en cours de planification
			NumSegments	UINT	Nombre de segments du chemin
			Segments	MATRICE d'UINT	Matrice de segments
0x02	Requise	Get/Set	TimeOut	UDINT	Délai d'attente du chien de garde (µs) (valeur par défaut = 60 s)
0x03	Facultatif	Get/Set	État du contrôleur	UINT	bit 0 (exécution/programme) = 0, peut être programmé = 1, contrôle le bit d'E-S bit 1 (distant/local) = 0, le bit 0 ne doit pas être modifié à distance = 1, le bit 0 peut être modifié à distance bits 2 à 15 réservés et doivent être nuls

7.4.4 Services communs

7.4.4.1 Généralités

Outre les services décrits dans le présent paragraphe, l'objet de planification doit prendre en charge le Get_Attribute_All standard adressé à l'instance de classe (instance zéro). La mise en œuvre d'un objet de planification particulier peut éventuellement prendre en charge d'autres attributs de service standard.

L'objet de planification doit prendre en charge les services communs spécifiés dans le Tableau 36.

Tableau 36 – Services communs de l'objet de planification

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service
	Classe	Instance		
0x01	Requise	Facultatif	Get_Attribute_All	Renvoie une liste prédéfinie des attributs de cet objet. Modifie tous les attributs définissables
0x02	N/A	Facultatif	Set_Attribute_All	Modifie tous les attributs définissables
0x03	Facultatif	Facultatif	Get_Attribute_List	Renvoie une liste d'attributs spécifiés
0x04	N/A	Facultatif	Set_Attribute_List	Modifie une liste d'attributs définissables
0x08	Requise	Facultatif	Create	Permet d'instancier un objet de planification
0x09	N/A	Requise	Delete	Permet de mettre fin à une session de planification LSS
0x0E	Facultatif	Facultatif	Get_Attribute_Single	Renvoie le contenu de l'attribut spécifié
0x10	N/A	Facultatif	Set_Attribute_Single	Modifie un seul attribut

Une mise en œuvre peut prendre en charge la définition de l'attribut au niveau de l'instance de la route, mais si ce n'est pas le cas, il ne doit pas être possible de distinguer le comportement interne d'un service delete suivi d'un service create contenant la nouvelle route.

7.4.4.2 Create

Le service Create doit être utilisé pour instancier un objet de planification sur le CO. Dans le message Create, le client (LSS) doit inclure une route pointant du CO vers la liaison à planifier. La route vers la liaison doit être composée de segments de port et ne doit pas contenir de segments de réseau. L'objet de planification doit répondre par un numéro d'instance auquel toutes les communications ultérieures doivent être adressées pendant cette session de planification, et par les mots de passe et chemin CO déjà enregistrés.

La nouvelle instance doit se supprimer elle-même si les communications avec son client sont coupées. Les communications avec une instance doivent être surveillées à l'aide d'un temporisateur dont la valeur de délai d'attente (par défaut = 60 s) peut être attribuée par l'un des services d'attribut de définition facultatifs. Le temporisateur d'une instance doit être réinitialisé lorsqu'il reçoit un message. Si le temporisateur expire, l'instance doit être supprimée en annulant toutes les modifications en attente. La valeur de délai d'attente doit être un entier de 32 bits exprimé en µs.

Si le service Create s'adresse au niveau de classe (instance zéro), le numéro d'instance de la nouvelle instance créée doit être choisi automatiquement par l'objet de planification. Pour les besoins du débogage, l'objet de planification peut éventuellement permettre au service Create d'être adressé à un numéro d'instance particulier.

L'objet de planification doit prendre en charge au moins une instance. Si l'objet de planification admet l'existence de plusieurs instances, il doit s'assurer de l'intégrité de chacune d'elles. La valeur de l'attribut d'instance 0x01 doit être fournie avec le service Create.

```

class Scheduling_Create_Request : public MR_Request
{
    UINT    number_of_attributes;    // set to 1
    InstanceAttribute1 attribute1;
}

class Scheduling_Create_Response : public MR_Response
{
    UDINT    instance_number;
    UDINT    cop;                // Connection Originator Password (COP)
    UINT     path_size;          // in words, range 0 to 255
    UINT     path[];             // array of port segments from the scheduled link
                                // to the connection originator
                                // no key segments, no network segments
                                // include the MAC ID of the link hop
};
    
```

La réponse doit contenir l'un des codes d'état figurant dans le Tableau 37.

Tableau 37 – Descriptions de l'erreur d'état de Create

État général	État étendu	Description de l'erreur
0x02	0x0001	Ressources insuffisantes — le nombre maximal d'instances d'objet de planification existe déjà
	0x0002	Ressources insuffisantes — mémoire insuffisante sur le dispositif CO
0x04	N/A	Erreur de syntaxe du chemin
0x08	N/A	Service non mis en œuvre (les instances non nulles étant facultatives)
0x10	N/A	Impossible d'ouvrir une autre instance suite à des conflits internes
0x13	N/A	Données de demande insuffisantes — la demande est trop courte ou tronquée
0x0D	N/A	L'objet existe déjà (lorsqu'une instance non nulle est spécifiée)
0x1C	N/A	Pénurie de listes d'attributs — il manque l'attribut requis
0xD0	N/A	Aucune connexion planifiée sur cette liaison

7.4.4.3 Delete

Le service Delete doit être utilisé pour mettre fin à une session de planification du LSS. Il permet de désallouer toutes les ressources d'une instance spécifiée de l'objet de planification. Toutes les modifications en attente qui n'ont pas encore été engagées doivent être perdues. Les connexions rompues par des commandes antérieures dans la session LSS ne doivent pas nécessairement être restaurées. Le service Delete ne doit pas être pris en charge au niveau de la classe. La réponse de ce service doit être spécifiée conformément au Tableau 38.

Tableau 38 – Description d'erreur d'état pour Delete et Kick_Timer

État général	État étendu	Description de l'erreur
0x00	N/A	Réussite
0x04	N/A	Erreur de syntaxe du chemin
0x05	N/A	Instance non définie
0x08	N/A	Service non mis en œuvre (si dirigé vers l'instance 0)

7.4.5 Services spécifiques à la classe

7.4.5.1 Généralités

L'objet de planification doit prendre en charge les services spécifiques à la classe suivants spécifiés dans le Tableau 39.

Tableau 39 – Services spécifiques à la classe de l'objet de planification

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service
	Classe	Instance		
0x4B	N/A	Requise	Kick_Timer	Réinitialisation du temporisateur dans l'objet de planification
0x4C	N/A	Requise	Read	Lit les informations relatives aux connexions planifiées
0x4D	N/A	Requise	Conditional_Write	Réécrit les informations de planification sur le dispositif CO
0x4E	N/A	Facultatif	Forced Write	Réécrit les informations de planification sur le dispositif CO
0x4F	N/A	Requise	Change Start	Lance une modification des données de planification
0x50	N/A	Requise	Break Connections	Interrompt toutes les connexions affectées après de nouvelles données de planification
0x51	N/A	Requise	Change Complete	Indique que toutes les connexions planifiées ont été interrompues
0x52	Requise	N/A	Restart Connections	Interrompt toutes les connexions qui utilisent une liaison particulière

7.4.5.2 Kick_Timer

Le service kick timer doit réinitialiser le temporisateur dans une instance de l'objet de planification sans affecter l'état de cet objet. Le LSS doit émettre quatre commandes par période de délai d'attente. Par exemple, le LSS doit émettre une demande de service toutes les 15 s si la valeur de délai d'attente est de 60 s par défaut. Tous les autres services de l'objet de planification, dirigés vers cette instance de planification, doivent également réinitialiser le temporisateur de manière à utiliser le service Kick_Timer uniquement si le client (LSS) souhaite conserver une instance de l'objet de planification active alors qu'il n'a rien à dire.

Le temporisateur d'une instance doit être réinitialisé lorsqu'il reçoit un message. Toutefois, son compte à rebours ne doit pas démarrer tant que la réponse correspondante n'a pas été envoyée. Ceci ne doit permettre la mise en œuvre que d'une seule tâche. La réponse de ce service doit être spécifiée conformément au Tableau 38.

7.4.5.3 Read

Le LSS doit utiliser le service Read pour lire les informations relatives aux connexions planifiées que le dispositif CO souhaite créer sur une liaison particulière. Les informations de connexion doivent provenir d'un objet d'application dans le dispositif CO.

Une demande de service Read doit contenir un paramètre UDINT requis qui spécifie l'indice de connexion sur lequel démarrer la lecture. L'objet de planification doit répondre à un service Read avec autant d'informations de connexion que ne peut en contenir un Lpacket de réponse. Les indices de connexion d'un Lpacket doivent être classés dans l'ordre croissant (les intervalles doivent être admis). Le champ d'état étendu de la réponse doit indiquer s'il

existe des connexions dotées d'indices plus élevés dans le dispositif CO. Un client (LSS) doit continuer à soumettre des demandes **read** avec des indices de démarrage plus élevés tant qu'il n'a pas lu toutes les informations de connexion de la liaison choisie. Un état étendu 0 doit indiquer qu'il existe plus de connexions dans le dispositif. Un état étendu 1 doit indiquer que toutes les connexions ont été lues.

La réponse doit être composée d'un UINT indiquant le nombre de connexions décrites dans cette réponse, suivi d'une série d'entrées décrivant chacune une connexion répondant aux critères de demande de connexion (la route vers la liaison). Chaque entrée doit être associée à un indice de données de planification unique, choisi et devant être utilisé principalement par le dispositif CO. Le dernier de ces indices peut être incrémenté par le LSS pour poursuivre la demande si elle est trop longue pour une réponse.

Les autres paramètres fournis dans l'entrée de connexion doivent décrire les tailles de connexion, les types de connexion (multidiffusion/point à point) et les vitesses de mise à jour des connexions (RPI). Les informations relatives à la route d'entrée de connexion doivent être composées de la route entre la liaison et le module cible pour la connexion comprenant les points de connexion du module cible. Il n'est pas utile que le LSS utilise réellement les informations relatives au port.

Cette route doit également contenir les valeurs en cours attribuées à la connexion dans les segments de réseau. Les segments de réseau doivent contenir les informations relatives à l'API (intervalle Lpacket) et à la NUT de départ.

```
class Scheduling_Read_Request : public MR_Request
{
    UDINT first_connection_index;
}

class Scheduling_Read_Response : public MR_Response
{
    UINT number_of_connections;
    Scheduling_Connection_Description desc[];
}

class Scheduling_Connection_Description
{
    UDINT connection_index;
    UINT O2T_parameters;
    UINT T2O_parameters;
    UDINT O2T_RPI;
    UDINT T2O_RPI;
    UINT path_size; // in words, range 0 to 255
    UINT path[]; // array of path segments
                // first segment is always O=>T schedule segment
                // second is always port segment onto the link
                // third is always T=>O schedule segment
                // remaining path including logical segments
                // and other port segments if multihop connection
};
```

L'outil de planification LSS doit utiliser les segments logiques du chemin cible afin de déterminer si plusieurs demandes de connexion utilisent réellement un producteur multipoint. La règle concernant tous les objets (autre l'objet Assembly) doit stipuler qu'une correspondance de classe, d'instance, de point de connexion $T \Rightarrow O$ et de granularité plus fine (attribut ou membre, par exemple) constitue un producteur unique et doit être planifiée une seule fois pour les deux connexions. Étant donné que l'objet Assembly compare une instance et un point de connexion, une correspondance d'une instance de chemin de connexion ne doit pas déterminer le caractère unique du producteur.

Par exemple, ces deux chemins de connexion spécifient le même producteur (instance 0x88 de l'objet Assembly):

"20 04 24 03 2C 99 2C 88";

"20 04 24 AA 2C 77 2C 88".

Ces deux chemins de connexion spécifient des producteurs différents, étant donné que le segment de classe logique ne correspond pas:

"20 77 24 03 2C 99 2C 88";

"20 77 24 AA 2C 99 2C 88".

La réponse d'état de ce service doit être spécifiée conformément au Tableau 40.

Tableau 40 – Descriptions de l'erreur d'état de Read

État général	État étendu	Description de l'erreur
0x00	N/A	Réussite
0x04	N/A	Erreur de syntaxe du chemin
0x05	N/A	Instance non définie
0x08	N/A	Service non mis en œuvre (si dirigé vers l'instance 0)
0x13	N/A	Données de demande insuffisantes — la demande est trop courte ou tronquée

7.4.5.4 Conditional_Write

Le service Conditional_Write doit être utilisé par le LSS pour réécrire les informations de planification dans le dispositif CO. La réception de ce message doit indiquer que le LSS planifie de nouveau des indices particuliers des informations de connexion. Un dispositif CO peut commencer par interrompre les connexions en cours, associées aux indices de connexion ou attendre la réception d'une demande de service d'interruption des connexions consécutive.

Si la nouvelle planification fournie par le LSS correspond exactement pour une connexion donnée, le dispositif CO peut ignorer cette partie de la demande d'écriture. Une mise en œuvre de l'objet de planification peut répondre par une erreur de ressource indisponible si elle reçoit plus de mises à jour que demandées pour plusieurs connexions (par l'intermédiaire de demandes Read). Une mise en œuvre peut également vérifier les demandes d'écriture en double et donc autoriser les nouvelles tentatives (plus de demandes d'écriture que de lecture). Les indices de connexion peuvent ne pas être dans l'ordre.

La différence entre le service d'écriture conditionnelle et le service d'écriture forcée doit être que, dans le premier cas, le dispositif CO doit être requis pour vérifier la fraîcheur des informations de connexion préalablement envoyées au LSS. Si les données ne sont pas fraîches, le dispositif CO ne doit pas utiliser les données de planification pour cet indice de connexion. Dans le cas du service d'écriture forcée, il n'est pas utile que le dispositif CO procède à une vérification. Informations de connexion « fraîches » s'entend qu'elles n'ont pas été modifiées depuis leur envoi au LSS à l'aide de la commande Read.

La demande doit être une matrice de planifications de connexion.

```
class Scheduling_Write_Request : public MR_Request
{
    UINT    number_of_connection_schedules;
    struct {
        UDINT connection_index;
        USINT  O2T_schedule;
        USINT  T2O_schedule;
    } schedules[];
};
```

L'état de la réponse de ce service doit être spécifié conformément au Tableau 41.

Tableau 41 – Descriptions de l'erreur d'état de Conditional_Write

État général	État étendu	Description de l'erreur
0x00	N/A	Réussite
0x03	N/A	Valeur non valide — La requête contient un indice incorrect
0x04	N/A	Erreur de syntaxe du chemin
0x05	N/A	Instance non définie
0x08	N/A	Service non mis en œuvre (si dirigé vers l'instance 0)
0x0C	N/A	État incorrect (Change_Start non reçu)
0x10	N/A	Conflit d'état du dispositif (le dispositif ne peut pas interrompre la connexion)
0x13	N/A	Données de demande insuffisantes — la demande est trop courte ou tronquée
0x15	N/A	L'instance n'est pas en mesure de recevoir d'autres demandes d'écriture

7.4.5.5 Forced_Write

Le service Forced_Write doit être analogue au service Conditional_Write, sauf que le LSS doit garantir avoir fourni des informations de connexion valides. Les paramètres de demande doivent être identiques à ceux du service Conditional_Write. L'état de la réponse de ce service doit être spécifié conformément au Tableau 42.

Tableau 42 – Descriptions de l'erreur d'état de Forced_Write

État général	État étendu	Description de l'erreur
0x00	N/A	Réussite
0x04	N/A	Erreur de syntaxe du chemin
0x05	N/A	Instance non définie
0x08	N/A	Service non mis en œuvre (si dirigé vers l'instance 0)
0x0C	N/A	État incorrect (Change_Start non reçu)
0x10	N/A	Conflit d'état du dispositif (le dispositif ne peut pas interrompre la connexion)
0x13	N/A	Données de demande insuffisantes — la demande est trop courte ou tronquée
0x15	N/A	L'instance n'est pas en mesure de recevoir d'autres demandes d'écriture

7.4.5.6 Change_Start

Le service Change_Start doit être utilisé pour lancer une modification de données de planification dans le contrôleur. Le contrôleur doit allouer suffisamment de mémoire pour les nouvelles données, le mot de passe et le chemin CO. Le service **Conditional_Write** ou **Forced_Write** doit être utilisé pour écrire les données dans le contrôleur.

Le mot de passe CO doit être utilisé lorsqu'un contrôleur est activé, puis commence à rétablir les connexions planifiées. Le CO doit s'assurer que le chemin associé au mot de passe CO (voir réponse Create ou demande Change_Complete) correspond à celui du chemin entre le réseau et le dispositif CO. S'il correspond, le CO doit s'assurer que ce mot de passe CO correspond à celui stocké dans l'objet Keeper de la liaison, avant d'établir des connexions planifiées sur ladite liaison.

La demande Change_Start ne doit comporter qu'un seul paramètre. Le paramètre doit être un UINT compris entre 0 et 255 qui spécifie la taille du mot de passe et du chemin CO à fournir dans le service Change_Complete. Les mises en œuvre peuvent l'utiliser pour allouer une mémoire tampon de réception pour le mot de passe et le chemin CO.

L'état de la réponse de ce service doit être spécifié conformément au Tableau 43.

Tableau 43 – Descriptions de l'erreur d'état de Change_Start

État général	État étendu	Description de l'erreur
0x00	N/A	Réussite
0x02	0x0002	Ressources insuffisantes — mémoire insuffisante sur le dispositif CO
0x04	N/A	Erreur de syntaxe du chemin
0x05	N/A	Instance non définie
0x08	N/A	Service non mis en œuvre (si dirigé vers l'instance 0)
0x10	N/A	Conflit d'état du dispositif (le dispositif ne peut pas allouer suffisamment de mémoire, par exemple)
0x13	N/A	Données de demande insuffisantes — la demande est trop courte ou tronquée

7.4.5.7 Break_Connections

Le service Break_Connections doit être utilisé après avoir écrit les nouvelles données de planification dans le contrôleur afin d'interrompre toutes les connexions concernées. Le service doit répondre uniquement après l'interruption de toutes les connexions. Si des connexions sont en cours, elles doivent également être interrompues avant de répondre à ce service.

Le service Break_Connections de l'objet de planification doit interrompre toutes les connexions spécifiées dans les services d'écriture de données de planification. Un LSS peut toujours écrire des données de planification pour toutes les connexions (interrompant toutes les connexions sur la liaison); mais un autre LSS peut être optimisé pour limiter le nombre de connexions interrompues.

Lors de la réception du service Break_Connections de l'objet de planification, outre l'interruption des connexions modifiées, le CO doit attribuer une valeur nulle (zéro) aux données planifiées actives pour les connexions qu'il interrompt, afin éviter de les rétablir tant que le service Change_Complete des données de planification n'a pas été reçu. Si le service Change_Complete n'est jamais reçu (les communications avec le LSS ayant été interrompues), ces connexions ne doivent pas pouvoir être ouvertes tant qu'elles n'ont pas été de nouveau planifiées. La réponse à ce service doit être spécifiée conformément au Tableau 44.

Tableau 44 – Descriptions de l'erreur d'état de Break_Connections

État général	État étendu	Description de l'erreur
0x00	N/A	Réussite
0x04	N/A	Erreur de syntaxe du chemin
0x05	N/A	Instance non définie
0x08	N/A	Service non mis en œuvre (si dirigé vers l'instance 0)
0x0C	N/A	Conflit d'état de l'objet
0x10	N/A	Conflit d'état du dispositif (le dispositif ne peut pas interrompre les connexions)
0x13	N/A	Données de demande insuffisantes — la demande est trop courte ou tronquée

7.4.5.8 Change_Complete

Le service Change_Complete doit être utilisé pour indiquer que toutes les connexions planifiées ont été interrompues et qu'il est admis d'établir les nouvelles connexions planifiées. La demande contient les nouveaux mot de passe et chemin CO utilisés pour la liaison qui a été planifiée. Le CO doit s'assurer que le chemin associé au mot de passe CO (voir réponse Create ou demande Change_Complete) correspond à celui du chemin entre le réseau et le dispositif CO. S'il correspond, le CO doit s'assurer que ce mot de passe CO correspond à celui stocké dans l'objet Keeper de la liaison, avant d'établir des connexions planifiées sur ladite liaison.

```
class Scheduling_Change_Complete_Request : public MR_Request
{
    UDINT cop;           // Connection Originator Password (COP)
    UINT  path_size;      // in words, range 0 to 255
    UINT  path[];         // array of port segments from the scheduled link
                        // to the connection originator
                        // no key segments, no network segments
                        // include the MAC ID of the link hop
};
```

NOTE Le chemin inverse de cette demande permet au LSS de détecter les modifications dans la position de réseau du CO planifié pendant les dernières sessions de planification.

L'état de la réponse de ce service doit être spécifié conformément au Tableau 45.

Tableau 45 – Descriptions de l'erreur d'état de Change_Complete

État général	État étendu	Description de l'erreur
0x00	N/A	Réussite
0x03	N/A	Valeur non valide (les nouveaux mot de passe et chemin CO ne sont pas correctement mis en forme)
0x04	N/A	Erreur de syntaxe du chemin
0x05	N/A	Instance non définie
0x08	N/A	Service non mis en œuvre (si dirigé vers l'instance 0)
0x10	N/A	Conflit d'état du dispositif
0x13	N/A	Données de demande insuffisantes — la demande est trop courte ou tronquée
0x0C	N/A	État incorrect

7.4.5.9 Restart_Connections

Le service Restart_Connections peut uniquement être envoyé à l'instance de classe de l'objet de planification (instance zéro). Il doit être utilisé pour interrompre toutes les connexions qui utilisent une liaison particulière. Suite à l'interruption de toutes les connexions, le dispositif CO doit tenter de les rétablir. Le paramètre de demande de ce service doit être structuré comme l'attribut d'instance 0x01. L'état de la réponse de ce service doit être spécifié conformément au Tableau 46.

Tableau 46 – Descriptions de l'erreur d'état de Restart_Connections

État général	État étendu	Description de l'erreur
0x00	N/A	Réussite
0x04	N/A	Erreur de syntaxe du chemin
0x08	N/A	Service non mis en œuvre (s'il est adressé à une instance non nulle)
0x10	N/A	Conflit d'état du dispositif (impossible d'interrompre les connexions)
0x13	N/A	Données de demande insuffisantes — la demande est trop courte ou tronquée
0xD0	N/A	Aucune connexion planifiée sur cette liaison

7.4.6 Session de planification classique

Une session de planification classique doit utiliser ces services:

- a) Create;
- b) Read;
- c) Write;
- d) Change_Start;
- e) Break_Connections;
- f) Conditional_Write;
- g) Forced_Write;
- h) Change_Complete.

La procédure ci-dessous décrit l'utilisation des services dans une session de planification classique:

- 1) Le LSS doit commencer à envoyer un message **create** à une instance zéro de l'objet de planification. Ce message doit contenir un chemin entre le CO et la liaison. Ce chemin doit déterminer les connexions présentant un intérêt pour le LSS.
- 2) L'objet de planification doit créer une instance de lui-même au service de ce LSS et répondre par le numéro d'instance auquel toutes les communications ultérieures doivent être envoyées.
- 3) L'instance qui vient d'être créée doit réinitialiser un temporisateur de 60 s dès la réception d'un message. Si ce temporisateur expire, l'instance doit se supprimer elle-même en annulant toutes les modifications en cours. En fonctionnement normal, le LSS doit envoyer une commande à l'objet de planification au moins une fois toutes les 15 s, en permettant de supprimer quelques messages sans supprimer inutilement une instance.
- 4) Le LSS peut éventuellement envoyer des commandes **read** à l'instance qui vient d'être créée afin de transférer les informations de connexion depuis le dispositif CO. Les informations de connexion transmises doivent contenir les indices utilisés pour identifier de manière unique les connexions en interne dans le dispositif CO.

- 5) Le LSS doit refléter ces indices sur des commandes **write** consécutives, de sorte que le CO puisse déterminer les connexions auxquelles s'applique la commande **write**. L'émission de commandes **read** doit être facultative, de manière à pouvoir obtenir par d'autres moyens les informations de connexion pour le LSS. Par exemple, dans un LSS/PS intégré, le PS doit fournir ces informations au LSS.
- 6) Après avoir obtenu les informations de connexion auprès de chaque CO souhaitant planifier des données sur la liaison, le LSS doit calculer une planification provisoire.
- 7) Le LSS doit alors émettre la séquence de commandes suivante vers chaque dispositif CO: **Change_Start**, un ou plusieurs **Write** et **Break_Connections**. Cette séquence peut être réalisée parallèlement à chacun des dispositifs CO. Le type de commande **Write** (forcée ou conditionnelle) doit dépendre de la manière dont les informations de connexion ont été obtenues. Si le LSS a indépendamment assuré que les informations de connexion sont valides, il peut utiliser la commande **Forced_Write**. Sinon, il doit utiliser une commande **Conditional_Write**. Le LSS peut savoir que les informations de connexion sont valides en obtenant la ressource d'édition du CO avant les informations de connexion.
- 8) Dès la réception de la commande **Conditional_Write**, l'objet de planification doit vérifier que les paramètres de la connexion n'ont pas été modifiés depuis la dernière lecture. Si la connexion a été modifiée, l'objet de planification ne doit pas utiliser les informations de planification pour l'indice dont les paramètres de connexion ont été modifiés. L'objet de planification du CO peut le faire par tous les moyens dont il dispose.
- 9) L'objet de planification doit être déchargé de toute responsabilité de vérification de la validité des commandes **Forced_Write**. La responsabilité doit être assumée par le LSS.
- 10) Lorsque les commande **Break_Connection** destinées à chacun des dispositifs CO ont abouti, le LSS doit émettre une commande **Change_Complete** destinée à chaque dispositif CO en indiquant que la planification préalablement transmise via les commandes **Write** est à présent valide. La commande **Change_Complete** doit inclure les nouveaux mot de passe et chemin CO.
- 11) La session doit se terminer par la suppression de l'instance de l'objet de planification.

7.5 Objet d'interface TCP/IP

7.5.1 Présentation

L'objet d'interface TCP/IP fournit le mécanisme de configuration de l'interface TCP/IP d'un dispositif.

NOTE 1 Des exemples d'éléments configurables incluent l'adresse IP du dispositif, le masque de réseau et l'adresse de passerelle.

Le port physique associé à l'objet d'interface TCP/IP doit être un port prenant en charge le protocole TCP/IP.

NOTE 2 Par exemple, un objet d'interface TCP/IP peut être associé à: un port Ethernet ISO/CEI 8802-3, un port ATM, un port série exécutant SLIP, un port série exécutant PPP.

L'objet d'interface TCP/IP offre un attribut qui identifie l'objet spécifique à la liaison pour le port physique associé. En règle générale, l'objet spécifique à la liaison est censé fournir des compteurs et des attributs de configuration spécifiques à la liaison.

Chaque dispositif doit prendre en charge exactement une instance de l'objet d'interface TCP/IP pour chaque port TCP/IP qu'il contient. Une demande adressée à l'instance d'accès 1 de l'objet d'interface TCP/IP doit toujours faire référence à l'instance associée au port sur lequel la demande a été reçue.

7.5.2 Attributs de classe

L'objet d'interface TCP/IP doit prendre en charge les attributs de classe spécifiés dans le Tableau 47.

Tableau 47 – Attributs de classe de l'objet d'interface TCP/IP

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x01	Facultatif	Get	Revision	UINT	Révision de cet objet	Première révision, valeur = 1

7.5.3 Attributs d'instance

7.5.3.1 Généralités

L'objet d'interface TCP/IP doit prendre en charge les attributs d'instance spécifiés dans le Tableau 48.

Tableau 48 – Attributs d'instance de l'objet d'interface TCP/IP

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x01	Requise	Get	État	DWORD	État de l'interface	Voir 7.5.3.2
0x02	Requise	Get	Capacité de configuration	DWORD	Balises de capacité d'interface	Matriciel de balises de capacité. Voir 7.5.3.3
0x03	Requise	Get/Set	Contrôle de configuration	DWORD	Balises de contrôle d'interface	Matriciel de balises de contrôle. Voir 7.5.3.4
0x04	Requise	Get	Objet de liaison physique	STRUCT de taille variable	Chemin vers l'objet de liaison physique	Voir 7.5.3.5
			Taille du chemin	UINT	Taille du chemin	Nombre d'UINT dans le chemin
			Chemin	MATRICE d'UINT	Segments logiques identifiant l'objet de liaison physique	Segment de classe et segment d'instance. La longueur maximale est de 6 UINT (si des segments logiques de 32 bits sont utilisés)
0x05	Requise	Get/Set	Configuration d'interface	STRUCT de taille variable	Configuration de l'interface réseau TCP/IP	Voir 7.5.3.6
			Adresse IP	UDINT	Adresse IP du dispositif.	Une valeur 0 indique qu'aucune adresse IP n'a été configurée. Sinon, l'adresse IP doit être définie sur une classe A, B ou C valide et ne doit l'être sur l'adresse en boucle (127.0.0.1)
			Masque de réseau	UDINT	Masque de réseau du dispositif	Une valeur 0 indique qu'aucun masque de réseau n'a été configuré

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
			Adresse de passerelle	UDINT	Adresse de la passerelle	Une valeur 0 indique qu'aucune adresse IP n'a été configurée. Sinon, l'adresse IP doit être définie sur une classe A, B ou C valide et ne doit l'être sur l'adresse en boucle (127.0.0.1)
			Serveur de noms	UDINT	Serveur de noms principal	Une valeur 0 indique qu'aucune adresse de serveur de noms n'a été configurée. Sinon, l'adresse de nom de serveur doit être définie sur une adresse de classe A, B ou C valide
			Serveur de noms 2	UDINT	Serveur de noms secondaire	Une valeur 0 indique qu'aucune adresse de serveur de noms secondaire n'a été configurée. Sinon, l'adresse de nom de serveur doit être définie sur une adresse de classe A, B ou C valide
			Nom de domaine	STRING	Nom de domaine par défaut	Caractères ASCII. La longueur maximale est de 48 caractères. Doit être rempli jusqu'à un nombre pair de caractères (remplissage exclu de la longueur). Une longueur de 0 doit indiquer qu'aucun nom de domaine n'est configuré
0x06	Requise	Get Set est facultative (voir 7.5.3.7)	Nom d'hôte	STRING	Nom d'hôte	Caractères ASCII. La longueur maximale est de 64 caractères. Doit être rempli jusqu'à un nombre pair de caractères (remplissage exclu de la longueur). Une longueur de 0 doit indiquer qu'aucun nom d'hôte n'est configuré. Voir 7.5.3.7
0x07	Conditionnel ^a		Numéro de réseau de sécurité	OCTET[6]		Voir CEI 61784-3-2
0x08	Conditionnel ^b	Get Set est conditionnel ^c	Valeur TTL	USINT	Valeur TTL de paquets multidiffusion CP 2/2	Valeur de durée de vie des paquets multidiffusion IP. La valeur par défaut est 1, la valeur minimale est 1, la valeur maximale est 255. Voir 7.5.3.8

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x09	Conditionnel ^b	Get Set est conditionnel ^c	Config Mcast	STRUCT de 8 octets	Configuration de l'adresse multidiffusion IP	Voir 7.5.3.9
			Alloc control	USINT	Mot de contrôle d'allocation d'adresse multidiffusion. Détermine la manière d'allouer les adresses	Détermine si des adresses multidiffusion sont générées via un algorithme ou sont explicitement définies. Voir 7.5.3.9 pour plus de détails
			Réservé	USINT	Réservé pour une utilisation future	Doit être nul
			Num mcast	UINT	Nombre d'adresses multidiffusion IP à allouer pour CP 2/2	Le nombre d'adresses multidiffusion IP alloués, commençant à « Mcast start addr ». La valeur maximale est spécifique au dispositif. Toutefois, elle ne doit pas dépasser le nombre de connexions multidiffusion CP 2/2 pris en charge par le dispositif
			Mcast start addr	UDINT	Démarrage de l'adresse multidiffusion à partir de laquelle commencer l'allocation	Adresse multidiffusion IP (Classe D). Un bloc d'adresses « Num mcast » alloué commençant avec cette adresse

^a Cet attribut est obligatoire pour les dispositifs de sécurité CPF 2. Les dispositifs non relatifs à la sécurité ne doivent pas mettre en œuvre cet attribut.

^b Si « valeur TTL » ou « Mcast config » est mis en œuvre, les deux doivent l'être.

^c Si « valeur TTL » ou « Mcast config » est mis en œuvre comme étant définissable, les deux doivent l'être en tant que tel.

7.5.3.2 Status

L'attribut Status est un matriciel indiquant l'état de l'interface réseau TCP/I (voir le Tableau 49. Consulter le diagramme d'états à la Figure 21 pour obtenir une description des états d'objet en fonction de leur attribut Status.

Tableau 49 – Bits Etat

Bit(s)	Nom	Définition
0-3	Interface configuration status	Indique l'état de l'attribut de configuration de l'interface. 0 = L'attribut de configuration de l'interface n'a pas été configuré 1 = L'attribut de configuration de l'interface contient une configuration valide obtenue auprès de BOOTP, DHCP ou de la mémoire non volatile 2 = L'attribut de configuration de l'interface contient une configuration valide, obtenue auprès des paramètres matériels (poussoir, molette,...) 3-15 = Réserve pour une utilisation future
4	Mcast pending	Indique une modification de configuration en attente dans la valeur TTL et/ou les attributs de configuration Mcast. Une valeur doit être attribuée à ce bit lorsque la valeur TTL ou l'attribut de configuration Mcast est défini, ce bit doit être supprimé au prochain démarrage du dispositif
5-31	Réserve	Réserve pour une utilisation future et doit être défini sur zéro

7.5.3.3 Capacité de configuration

L'attribut Capacité de configuration est un matriciel qui indique le support du dispositif pour une capacité de configuration facultative du réseau. Les éléments possibles de capacité de configuration figurent dans le Tableau 50. Il n'est pas indispensable que les dispositifs prennent en charge un élément particulier. Toutefois, ils doivent prendre en charge au moins une méthode permettant d'obtenir une adresse IP initiale.

Tableau 50 – Bits de capacité de configuration

Bit	Nom	Définition
0	BOOTP client	1 (TRUE) doit indiquer que le dispositif est en mesure d'obtenir sa configuration de réseau par l'intermédiaire de BOOTP
1	DNS client	1 (TRUE) doit indiquer que le dispositif est en mesure de résoudre les noms d'hôte en interrogeant un serveur DNS
2	DHCP client	1 (TRUE) doit indiquer que le dispositif est en mesure d'obtenir sa configuration de réseau par l'intermédiaire de DHCP
3	DHCP-DNS update	1 (TRUE) doit indiquer que le dispositif est en mesure d'envoyer son nom d'hôte dans la demande DHCP comme indiqué dans Internet draft <draft-ietf-dhc-fqdn-option-12.txt>
4	Configuration settable	1 (TRUE) doit indiquer qu'une valeur peut être allouée à l'attribut de configuration de l'interface. Certains dispositifs (un PC ou un poste de travail, par exemple) peuvent ne pas permettre de définir la configuration d'interface par l'intermédiaire de l'objet d'interface TCP/IP
5-31	Réserve	Réserve pour une utilisation future et doit être défini sur zéro

7.5.3.4 Contrôle de configuration

L'attribut Contrôle de configuration est un matriciel permettant de contrôler les options de configuration de réseau (voir le Tableau 51).

Tableau 51 – Bits de contrôle de configuration

Bit	Name	Definition	
0–3	Configuration au démarrage	Détermine comment le dispositif doit obtenir sa configuration initiale au démarrage	0 = Le dispositif doit utiliser la configuration de réseau déjà enregistrée (dans la mémoire non volatile ou par l'intermédiaire de commutateurs matériels, par exemple) 1 = Le dispositif doit obtenir sa configuration de réseau par l'intermédiaire de BOOTP. 2 = Le dispositif doit obtenir sa configuration de réseau par l'intermédiaire de DHCP au démarrage. 3-15 = Réserve pour une utilisation future
4	DNS Enable	Si 1 (TRUE), le dispositif doit résoudre les noms d'hôte en interrogeant un serveur DNS	
5–31	Réservé	Réservé pour une utilisation future et doit être défini sur zéro	

Les dispositifs ne sont pas tenus de prendre en charge l'une des valeurs particulières des bits Configuration au démarrage. Toutefois, un dispositif doit prendre en charge au moins une méthode permettant d'obtenir une configuration d'interface TCP/IP initiale. Par exemple, certains dispositifs bas de gamme peuvent choisir d'obtenir la configuration de l'interface réseau par l'intermédiaire de BOOTP ou DHCP uniquement. D'autres dispositifs peuvent souhaiter n'admettre la configuration initiale que sur un port série (ou autre), puis stocker la configuration dans la mémoire non volatile.

L'utilisation de BOOTP et/ou DHCP peut s'avérer inappropriée pour certains dispositifs. DHCP, en particulier, prend en charge les adresses IP attribuées de manière dynamique, ce qui peut permettre à un dispositif d'obtenir une adresse IP différente à chaque mise sous tension. Ce comportement n'est pas approprié pour les dispositifs ayant besoin d'adresses IP statiques.

Prêt à l'emploi, un dispositif peut souhaiter obtenir sa configuration initiale par des méthodes autres que BOOTP ou DHCP. Par exemple, le dispositif peut souhaiter obtenir sa configuration initiale sur un port série connexe ou un autre port de communication compatible avec le protocole de couche d'application de type 2 (en supposant que le dispositif soit équipé d'un autre port de communication). Dans ce cas, la valeur 0 doit être attribuée aux bits Startup configuration du dispositif, ses champs d'attributs de configuration d'interface étant nuls. Le dispositif doit alors attendre d'être configuré sur un autre port de communication.

Une fois le dispositif activé et la valeur 0 attribuée aux bits Startup configuration, une demande de définition de l'attribut de configuration d'interface doit permettre au dispositif de stocker le contenu de l'attribut Interface configuration dans la mémoire non volatile, si le dispositif la prend en charge. La valeur 0 ne doit pas être attribuée aux bits Startup configuration tant que l'attribut Interface Configuration n'a pas été préalablement défini. Sinon, la communication du dispositif sur le réseau pourrait devenir impossible.

7.5.3.5 Objet de liaison physique

Cet attribut permet d'identifier l'objet associé au port physique sous-jacent. Il existe deux composants pour cet attribut: une taille de chemin (en UINT) et un chemin. Le chemin doit contenir un segment de classe et un segment d'instance identifiant l'objet de port physique. La taille de chemin maximale est de 6 (dans le cas d'un segment logique de 32 bits pour chacune des classes de l'instance). Un exemple de chemin est présenté dans le Tableau 52.

Tableau 52 – Exemple de chemin

Chemin	Signification
[20][f6][24][01]	[20] = type de segment de classe à 8 bits; [f6] = classe d'objet de liaison Ethernet; [24] = type de segment d'instance à 8 bits; [01] = instance 1

D'une manière générale, l'objet de liaison physique lui-même gère les compteurs et les attributs de configuration spécifiques à la liaison. L'objet de liaison physique le plus probable est l'objet de liaison Ethernet (voir 7.6).

7.5.3.6 Configuration d'interface

Cet attribut contient les paramètres de configuration requis pour fonctionner comme un nœud TCP/IP. Pour éviter les configurations incomplètes ou incompatibles, les paramètres composant l'attribut de configuration d'interface ne peuvent pas être définis individuellement. Pour modifier l'attribut de configuration d'interface, il convient que l'utilisateur l'extrait en premier lieu, modifie les paramètres de son choix, puis le définisse.

L'objet d'interface TCP/IP doit appliquer la nouvelle configuration dès la fin du service de définition. Si la valeur des bits Startup configuration (attribut Configuration control) est nulle, la nouvelle configuration doit être stockée dans la mémoire non volatile. Le dispositif ne doit pas répondre au service de définition tant que les valeurs n'ont pas été stockées dans la mémoire non volatile.

Si la configuration initiale doit être obtenue par l'intermédiaire de BOOTP ou de DHCP, tous les composants de l'attribut de configuration d'interface doivent être nuls tant que la réponse BOOTP ou DHCP n'a pas été reçue. Dès la réception de la réponse BOOTP ou DHCP, l'attribut de configuration d'interface doit présenter la configuration obtenue par l'intermédiaire de BOOTP/DHCP.

Les dispositifs ne sont pas obligés de prendre en charge le service de définition. Certaines mises en œuvre (celles fonctionnant sur un PC ou un poste de travail, par exemple) n'ont pas besoin de prendre en charge la définition de la configuration d'interface réseau par l'intermédiaire de l'objet d'interface TCP/IP.

Les composants des attributs de configuration d'interface sont présentés dans le Tableau 53.

Tableau 53 – Composants de configuration d'interface

Nom	Signification
Adresse IP	Adresse IP du dispositif
Masque de réseau	Masque de réseau du dispositif. Le masque de réseau est utilisé lorsque le réseau IP a été partitionné en sous-réseaux. Le masque de réseau permet de déterminer si une adresse IP se trouve dans un autre sous-réseau
Adresse de passerelle	Adresse IP de la passerelle par défaut du dispositif. Si une adresse IP de destination se trouve sur un autre sous-réseau, les paquets sont transférés vers la passerelle par défaut pour être acheminés vers le sous-réseau de destination
Serveur de noms	Adresse IP du serveur de nom principal. Le serveur de noms permet de résoudre les noms d'hôte. Par exemple, ils peuvent se trouver dans un chemin de connexion de bus de terrain
Serveur de noms 2	Adresse IP du serveur de nom secondaire. Le nom de serveur secondaire est utilisé lorsque le nom de serveur principal n'est pas disponible ou qu'il n'est pas en mesure de résoudre un nom d'hôte
Nom de domaine	Nom de domaine par défaut. Le nom de domaine par défaut est utilisé pour résoudre les noms d'hôte qui ne sont pas totalement qualifiés. Par exemple, si le nom de domaine par défaut est « network.org » et que le dispositif souhaite résoudre un nom d'hôte « plc », le dispositif va tenter de résoudre le nom d'hôte en « plc.network.org »

7.5.3.7 Nom d'hôte

L'attribut Nom d'hôte contient le nom d'hôte du dispositif. L'attribut Nom d'hôte est utilisé lorsque le dispositif prend en charge la fonction DHCP-DNS Update et a été configuré pour utiliser DHCP au démarrage. Le mécanisme permet au client DHCP de transmettre son nom d'hôte au serveur DHCP. Le serveur DHCP met alors à jour les enregistrements DNS pour le compte du client.

NOTE Le mécanisme DHCP-DNS Update est spécifié dans Internet draft <draft-ietf-dhc-fqdn-option-12.txt>.

Il n'est pas utile de définir l'attribut Nom d'hôte pour que le dispositif fonctionne normalement. La valeur de l'attribut Nom d'hôte, s'il est configuré, doit être utilisée pour l'option FQDN de la demande DHCP. Si l'attribut Nom d'hôte n'a pas été configuré, le dispositif ne doit pas inclure l'option FQDN dans la demande DHCP.

Pour les dispositifs qui ne prennent pas en charge la fonction DHCP-DNS ou ne sont pas configurés pour utiliser DHCP, le nom d'hôte peut être utilisé pour des besoins informatifs.

L'accès de définition est facultatif lorsque l'attribut de configuration d'interface ne peut pas être défini. Certains dispositifs (un PC ou un poste de travail, par exemple) peuvent ne pas permettre de définir les attributs de configuration d'interface ou Nom d'hôte par l'intermédiaire de l'objet d'interface TCP/IP. Si la définition n'est pas mise en œuvre, une erreur 0x0E (l'attribut ne peut pas être défini) doit être renvoyée en réponse à la demande « Set_Attribute_Single ».

7.5.3.8 Valeurs TTL

La valeur TTL est celle que le dispositif doit utiliser pour le champ de durée de vie de l'en-tête IP lors de l'envoi de paquets CP 2/2 par multidiffusion IP. Par défaut, la valeur TTL doit être 1. La valeur maximale de TTL est 255. Les paquets à diffusion individuelle doivent utiliser la valeur TTL configurée pour la pile TCP/IP, et pas celle configurée dans cet attribut.

Une fois définie, la valeur TTL doit être sauvegardée dans la mémoire non volatile et prendre effet au démarrage suivant du dispositif. La définition de l'attribut Valeur TTL doit également permettre de définir le bit Mcast pending de l'attribut Status de l'interface, indiquant qu'il s'agit d'une configuration en attente. Si une nouvelle valeur TTL est en attente, la demande

Get_Attribute_Single ou Get_Attribute_All doit renvoyer la valeur en attente. Le bit Mcast pending doit être supprimé au démarrage suivant du dispositif.

NOTE Il convient que les utilisateurs soient très prudents lorsqu'ils attribuent une valeur supérieure à 1 à TTL, afin d'éviter qu'un trafic multidiffusion indésirable ne se propage sur le réseau. La CEI 61158-6-2 contient une discussion relative aux considérations de l'utilisateur sur la multidiffusion.

7.5.3.9 Mcast config

L'attribut Config Mcast contient la configuration des adresses multidiffusion IP du dispositif à utiliser pour les paquets multidiffusion CP 2/2. La structure de Config Mcast est composée de trois éléments: Alloc control, Num mcast et Mcast start addr.

- Alloc control permet de déterminer la manière dont le dispositif doit attribuer des adresses multidiffusion IP (par un algorithme, une définition explicite, etc.). Le Tableau 54 présente les détails de l'élément alloc control.

Tableau 54 – Valeurs d'alloc control

Valeur	Définition
0	Les adresses multidiffusion doivent être générées à l'aide de l'algorithme d'allocation par défaut spécifié dans la CEI 61158-6-2. Si cette valeur est attribuée à Set_Attribute_Single ou Set_Attribute_All, les valeurs de Num mcast et Mcast start addr de la demande Set_Attribute doivent être nulles
1	Les adresses multidiffusion doivent être attribuées en fonction des valeurs spécifiées dans Num mcast et Mcast start addr
2	Réservé

- Num mcast est le nombre d'adresses multidiffusion IP attribuées. Le nombre maximal d'adresses multidiffusion est spécifique au dispositif. Toutefois, il ne doit pas dépasser le nombre de connexions multidiffusion CP 2/2 pris en charge par le dispositif.
- Mcast start addr est l'adresse multidiffusion de départ à partir de laquelle les adresses Num mcast sont attribuées.

Une fois défini, l'attribut Config Mcast doit être sauvegardé dans la mémoire non volatile et prendre effet au démarrage suivant du dispositif. La définition de l'attribut Config Mcast doit également permettre de définir le bit Mcast pending de l'attribut Status de l'interface, indiquant qu'il s'agit d'une configuration en attente. Si une nouvelle valeur Config Mcast est en attente, la demande Get_Attribute_Single ou Get_Attribute_All doit renvoyer la valeur en attente. Le bit Mcast pending doit être supprimé au démarrage suivant du dispositif.

Si les adresses multidiffusion sont générées à l'aide de l'algorithme par défaut, Num mcast et Mcast start addr doivent renvoyer les valeurs générées par l'algorithme.

7.5.4 Services communs

7.5.4.1 Généralités

L'objet d'interface TCP/IP doit prendre en charge les services communs spécifiés dans le Tableau 55.

Tableau 55 – Services communs de l'objet d'interface TCP/IP

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service
	Classe	Instance		
0x01	Facultatif	Facultatif	Get_Attribute_All	Renvoie une liste prédéfinie des attributs de cet objet (voir la définition de réponse Get_Attribute_All dans 7.5.4.2)
0x02	N/A	Facultatif	Set_Attribute_All	Modifie tous les attributs définissables
0x0E	Conditionnel ^a	Requis	Get_Attribute_Single	Renvoie le contenu de l'attribut spécifié
0x10	N/A	Requis	Set_Attribute_Single	Modifie un seul attribut
^a Le service Get_Attribute_Single doit être mis en œuvre pour la classe si des attributs de classe sont mis en œuvre.				

7.5.4.2 Get_Attribute_All response

Pour les attributs de classe, étant donné qu'il n'existe qu'un seul attribut de classe, l'attribut de classe #1 doit être renvoyé.

Pour les attributs d'instance, les attributs doivent être renvoyés dans l'ordre numérique jusqu'au dernier attribut mis en œuvre. La réponse Get_Attribute_All doit être telle que spécifiée dans le Tableau 56.

Tableau 56 – Format de réponse Get_Attribute_All

ID attribut	Taille (octets)	Contenu
1	4	État
2	4	Capacité de configuration
3	4	Contrôle de configuration
4	2	Objet de liaison physique, taille de chemin
	Variable, 12 octets max	Objet de liaison physique, chemin (si la taille de chemin est non nulle)
5	4	Adresse IP
	4	Masque de réseau
	4	Adresse de passerelle
	4	Serveur de noms
	4	Serveur de noms secondaire
	2	Longueur du nom de domaine
	Variable, égale à la longueur du nom de domaine	Nom de domaine
6	1	Octet de remplissage uniquement si la longueur du nom de domaine est impaire
	2	Longueur du nom d'hôte
	Variable, égale à la longueur du nom d'hôte	Nom d'hôte
7	1	Octet de remplissage uniquement si la longueur du nom d'hôte est impaire
	6	Voir CEI 61784-3-2.
		Absent si l'attribut 7 n'est pas mis en œuvre. Doit être nul si des attributs supplémentaires supérieurs à l'ID attribut 7 sont inclus

ID attribut	Taille (octets)	Contenu
8	1	Valeur TTL. Absent si l'attribut 8 n'est pas mis en œuvre. Doit être nul si des attributs supplémentaires supérieurs à l'ID attribut 8 sont inclus
9	8	Config Mcast. Absent si l'attribut 9 n'est pas mis en œuvre. Doit être nul si des attributs supplémentaires supérieurs à l'ID attribut 9 sont inclus

Les longueurs de chemin d'objet de liaison physique, de nom de domaine et de nom d'hôte sont inconnues avant d'émettre la demande de service Get_Attribute_All. Les implémentateurs doivent se préparer à accepter une réponse contenant les tailles maximales de chemin d'objet de liaison physique (6 UINT), de nom de domaine (48 USINT) et de nom d'hôte (64 USINT).

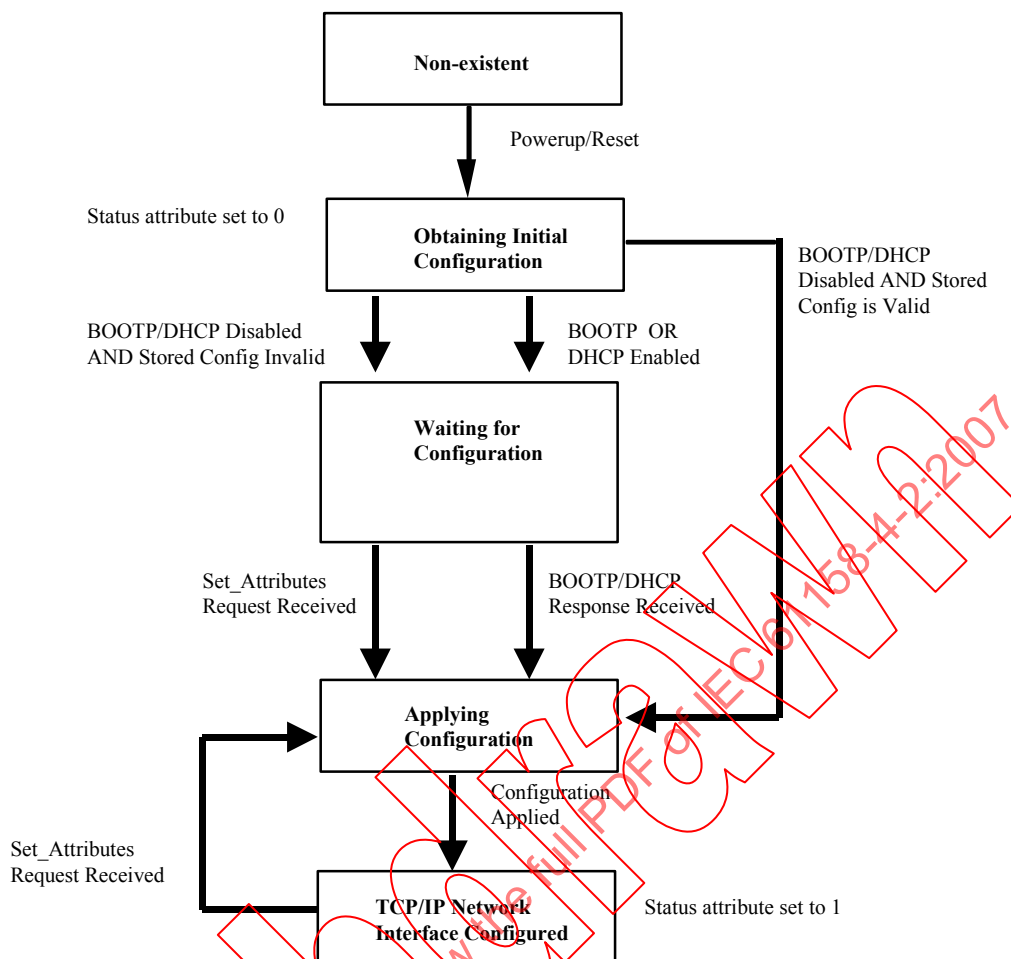
7.5.4.3 Demande Set_Attribute_All

La demande Set_Attribute_All de l'instance doit contenir l'attribut Contrôle de configuration, suivi de l'attribut de configuration d'interface.

7.5.5 Comportement

Le comportement de l'objet d'interface TCP/IP doit être tel qu'illustré dans le diagramme de transition d'états Figure 21.

A noter que l'obtention d'une image exécutable initiale par l'intermédiaire de BOOTP/TFTP ne doit pas être considérée dans le cadre du comportement de l'objet d'interface TCP/IP. Les dispositifs sont libres de mettre en œuvre ce type de comportement. Toutefois, il doit être considéré comme s'étant produit à l'état « inexistant ».



Légende

Anglais	Français
Non-existent	Non existant
Powerup/reset	Mise sous tension/réinitialisation
Status attribute set to 0	Attribut d'état mis à 0
Obtaining initial configuration	Obtention de la configuration initiale
BOOTP/DHCP disabled	BOOTP/DHCP désactivés
AND stored config is valid	ET config stockée valide
AND stored config invalid	ET config stockée invalide
BOOTP OR DHCP enabled	BOOTP OU DHCP activé
Waiting for configuration	Attente de la configuration
Set-attributes request received	Demande Set-attributes reçue
BOOTP/DHCP response received	Réponse BOOTP/DHCP reçue
Applying configuration	Application de la configuration
Configuration applied	Configuration appliquée
TCP/IP network interface configured	Interface réseau TCP/IP configurée
Status attribute set to 1	Attribut d'état mis à 1

Figure 21 – Diagramme de transition d'états de l'objet d'interface TCP/IP

7.6 Objet de liaison Ethernet

7.6.1 Présentation

L'objet de liaison Ethernet gère les compteurs spécifiques à la liaison et les informations d'état d'un port Ethernet physique ISO/CEI 8802-3. Chaque dispositif doit prendre en charge exactement une instance de l'objet de liaison Ethernet pour chaque port Ethernet du module. Une demande adressée à l'instance d'accès 1 de l'objet de liaison Ethernet doit toujours faire référence à l'instance associée au port sur lequel la demande a été reçue.

7.6.2 Historique de révision

L'historique de révision de l'objet de liaison Ethernet est décrit dans le Tableau 57.

Tableau 57 – Historique de révision de l'objet de liaison Ethernet

Révision de classe	Description des modifications
01	Remise initiale
02	Remise pour la série CEI 61158

7.6.3 Attributs de classe

L'objet de liaison Ethernet doit prendre en charge les attributs de classe spécifiés dans le Tableau 58.

Tableau 58 – Attributs de classe de l'objet de liaison Ethernet

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x01	Requise	Get	revision	UINT	révision de cet objet	valeur actuelle pour cet attribut = 2
0x02 à 0x07	Ces attributs de classe sont facultatifs et sont décrits dans la CEI 61158-5-2.					

Une erreur lors de la lecture de l'attribut de révision de classe implique qu'il s'agit d'une mise en œuvre de révision 1 uniquement.

7.6.4 Attributs d'instance

7.6.4.1 Généralités

L'objet de liaison Ethernet doit prendre en charge les attributs d'instance spécifiés dans le Tableau 59.

Tableau 59 – Attributs d'instance de l'objet de liaison Ethernet

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x01	Requise	Get	Vitesse d'interface	UDINT	Vitesse d'interface utilisée	Vitesse en Mbit/s (0, 10, 100, 1 000, par exemple). Voir 7.6.4.2

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x02	Requise	Get	Balises d'interface	DWORD	Balises d'état de l'interface	Matriciel de balises d'interface. Voir 7.6.4.3
0x03	Requise	Get	Adresse physique	USINT [6]	Adresse de la couche MAC	Voir 7.6.4.4
0x04	Conditionnel ^a	Get	Compteurs d'interface	STRUCT de 44 octets		Voir 7.6.4.5
			In Octets	UDINT	Octets reçus sur l'interface	
			In Ucast Packets	UDINT	Paquets à diffusion individuelle reçus sur l'interface	
			In NUCast Packets	UDINT	Paquets à diffusion non individuelle reçus sur l'interface	
			In Discards	UDINT	Paquets entrants reçus sur l'interface mais écartés	
			In Errors	UDINT	Paquets entrants contenant des erreurs (n'inclut pas In Discards)	
			In Unknown Protos	UDINT	Paquets entrants avec protocole inconnu	
			Out Octets	UDINT	Octets envoyés sur l'interface	
			Out Ucast Packets	UDINT	Paquets à diffusion individuelle envoyés sur l'interface	
			Out NUCast Packets	UDINT	Paquets à diffusion non individuelle envoyés sur l'interface	
			Out Discards	UDINT	Paquets sortants écartés	
			Out Errors	UDINT	Paquets sortants contenant des erreurs	
0x05	Facultatif	Get	Compteurs de support	STRUCT de 48 octets	Compteurs spécifiques au support	Voir 7.6.4.6
			Erreurs d'alignement	UDINT	Trames reçues dont la longueur n'est pas un nombre d'octets intégral	
			Erreurs FCS	UDINT	DLPDU reçues n'ayant pas satisfait au contrôle FCS	
			Collisions uniques	UDINT	DLPDU transmises avec succès ayant fait l'objet d'une seule collision	

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
			Collisions multiples	UDINT	DLPDU transmises avec succès ayant fait l'objet de plusieurs collisions	
			Erreurs d'essai SQE	UDINT	Nombre de fois qu'un message d'erreur d'essai SQE a été généré	
			Transmissions différées	UDINT	DLPDU dont la première tentative de transmission a été retardée car le support est occupé	
			Collisions tardives	UDINT	Nombre de fois qu'une collision est détectée après 512 temps bits dans la transmission d'un paquet	
			Collisions excessives	UDINT	DLPDU dont la transmission n'a pas abouti en raison de collisions excessives	
			Erreurs de transmission MAC	UDINT	DLPDU dont la transmission n'aboutit pas en raison d'une erreur de transmission de la sous-couche MAC	
			Erreurs de sens de porteuse	UDINT	Nombre de fois que le sens de la porteuse a été perdu ou n'a jamais été évaluée lors de la tentative de transmission d'une trame	
			Trame trop longue	UDINT	DLPDU reçues ayant dépassé la taille DLPDU maximale admise	
			Erreurs de réception MAC	UDINT	DLPDU dont la réception sur une interface n'a pas abouti en raison d'une erreur de réception de la sous-couche MAC interne	
0x06	Facultatif	Get/Set	Contrôle d'interface	STRUCT de	Configuration de l'interface physique	Voir 7.6.4.7
			Bits de contrôle	WORD	Bits de contrôle d'interface	
			Vitesse d'interface forcée	UINT	Vitesse à laquelle l'interface doit être forcée pour fonctionner	Vitesse en Mbit/s (10, 100, 1 000, par exemple).

^a L'attribut Compteurs d'interface est obligatoire si l'attribut Compteurs de support est mis en œuvre.

7.6.4.2 Vitesse d'interface

L'attribut Vitesse d'interface doit indiquer la vitesse de fonctionnement de l'interface (10 Mbit/s, 100 Mbit/s, 1 Gbit/s ou d'autres vitesses). Une valeur nulle doit être utilisée pour indiquer que la vitesse de l'interface est indéterminée.

L'échelle de l'attribut étant exprimée en Mbit/s, l'interface fonctionne à 100 Mbit/s. Par conséquent, la valeur de l'attribut Vitesse d'interface doit être 100. La vitesse de l'interface a pour objet de représenter la durée de transmission de liaison disponible. L'attribut ne doit pas être indiqué deux fois si l'interface fonctionne en mode bidirectionnel simultané.

7.6.4.3 Balises d'interface

L'attribut Balises d'interface contient les informations d'état et de configuration relatives à l'interface physique (voir le Tableau 60).

Tableau 60 – Bits des balises d'interface

Bit	Nom	Définition
0	État de la liaison	Indique si le port est connecté ou pas à un réseau actif: - 0 indique une liaison inactive - 1 indique une liaison active La détermination de l'état de la liaison est spécifique à la mise en œuvre. Dans certains cas, les dispositifs peuvent dire si la liaison est active par l'intermédiaire d'un support matériel/pilote. Dans d'autres cas, le dispositif peut uniquement être en mesure de dire si la liaison est active en présence de paquets entrants
1	Bidirectionnel non simultané/simultané	Indique le mode duplex utilisé: - 0 indique que le port fonctionne en mode duplex non simultané - 1 indique le mode duplex simultané Si la balise d'état de la liaison est 0, la valeur de la balise Bidirectionnel non simultané/simultané est indéterminée.
2-4	État de négociation	Indique l'état de l'auto-négociation de la liaison: 0 = Auto-négociation en cours 1 = Échec de l'auto-négociation et de la détection de la vitesse. Utilisation des valeurs par défaut pour la vitesse et le duplex. Les valeurs par défaut dépendent du produit. Les valeurs par défaut recommandées sont 10 Mbit/s et bidirectionnel non simultané 2 = L'auto-négociation n'a pas abouti mais la vitesse a été détectée. Duplex prend la valeur par défaut. La valeur par défaut dépend du produit. La valeur recommandée est bidirectionnel non simultané 3 = Vitesse et duplex négociés 4 = Auto-négociation non tentée. Vitesse et duplex forcés
5	Définition manuelle requiert réinitialisation	0 indique que l'interface peut activer automatiquement les modifications apportées aux paramètres de liaison (auto-négociation, mode duplex, vitesse d'interface) 1 indique que le dispositif implique d'émettre un service Reset pour son objet d'identification afin que les modifications entrent en vigueur
6	Défaut matériel local	0 indique que l'interface ne détecte aucun défaut matériel 1 indique qu'un défaut matériel local est détecté Sa signification est spécifique au produit. Par exemple: une interface AUI/MII ne détecte aucun émetteur-récepteur associé ou le modem cellulaire ne détecte aucune antenne associée. À l'inverse, la nature auto-correctrice de l'état de la liaison étant inactive, elle est considérée comme un incident machine impliquant l'intervention de l'utilisateur
7-31	Réservé	Réservé et une valeur nulle doit être attribuée.

7.6.4.4 Adresse physique

L'attribut Adresse physique contient l'adresse de la couche MAC de l'interface. L'adresse physique est une matrice d'octets. Le format d'affichage recommandé est « XX-XX-XX-XX-XX-XX », en commençant par le premier octet. L'adresse physique n'est pas un attribut définissable: l'adresse Ethernet doit être attribuée par le fabricant et doit être unique conformément aux exigences de l'ISO/CEI 8802-3.

7.6.4.5 Compteurs d'interface

L'attribut Compteurs d'interface contient tous les compteurs ayant rapport à la réception des paquets sur l'interface. Ces compteurs doivent être définis conformément au RFC 1213 "MIB-II Management Information Base". L'attribut Compteurs d'interface est conditionnel. Il doit être mis en œuvre si l'attribut Compteurs de support l'est également.

7.6.4.6 Compteurs de support

L'attribut Compteurs de support contient les compteurs spécifiques au support Ethernet. Ces compteurs doivent être définis conformément au RFC 1643, "Definitions of Managed Objects for Ethernet-Like Interface Types". Si cet attribut est mis en œuvre, l'attribut Compteurs d'interface doit l'être également.

7.6.4.7 Contrôle d'interface

7.6.4.7.1 Présentation

L'attribut Contrôle d'interface est une structure composée de bits de contrôle et d'une vitesse d'interface forcée, et doit être conforme aux spécifications des paragraphes ci-après.

7.6.4.7.2 Bits de contrôle

L'attribut Bits de contrôle est spécifié dans le Tableau 61.

Tableau 61 – Bits de contrôle

Bit	Nom	Définition
0	Auto-négociation	0 indique que l'auto-négociation de liaison ISO/CEI 8802-3 est désactivée 1 indique que l'auto-négociation est activée Si l'auto-négociation est désactivée, le dispositif doit utiliser les paramètres indiqués par le mode duplex forcé et les bits de vitesse d'interface forcée
1	Mode duplex forcé	Si la valeur 0 est attribuée au bit d'auto-négociation, le bit de mode duplex forcé indique si l'interface doit fonctionner en mode duplex simultané ou non simultané. 0 indique qu'il convient que l'interface fonctionne en mode duplex non simultané 1 indique qu'il convient que l'interface fonctionne en mode duplex simultané Les interfaces qui ne prennent pas en charge le mode duplex demandé doivent retourner un code d'erreur 0x09 (valeur d'attribut non valide). Si l'auto-négociation est activée, la tentative de définition des bits du mode duplex forcé doit donner un code d'erreur 0x0C (Conflit d'état de l'objet)
2-15	Réservé	Doit être nul

7.6.4.7.3 Vitesse d'interface forcée

Si la valeur 0 est attribuée au bit d'auto-négociation, les bits de vitesse d'interface forcée indiquent la vitesse à laquelle l'interface doit fonctionner. La vitesse est exprimée en Mbit/s

(par exemple, pour 10 Mbit/s Ethernet, la vitesse d'interface doit être 10). Il convient que les interfaces qui ne prennent pas en charge la vitesse demandée retournent un code d'erreur 0x09 (valeur d'attribut non valide).

Si l'auto-négociation est activée, la tentative de définition de la vitesse d'interface forcée doit donner un code d'erreur 0x0C (Conflit d'état de l'objet).

7.6.5 Services communs

7.6.5.1 Généralités

L'objet de liaison Ethernet doit prendre en charge les services communs spécifiés dans le Tableau 62.

Tableau 62 – Services communs de l'objet de liaison Ethernet

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service
	Classe	Instance		
0x01	Facultatif	Facultatif	Get_Attribute_All	Renvoie une liste prédéfinie des attributs de cet objet (voir la définition de réponse Get_Attribute_All dans 7.6.5.2)
0x0E	Sous condition	Requise	Get_Attribute_Single	Renvoie le contenu de l'attribut spécifié
0x10	N/A	Sous condition	Set_Attribute_Single	Modifie un seul attribut

Le service Get_Attribute_Single doit être mis en œuvre pour l'attribut de classe si ce dernier est mis en œuvre.

Le service Set_Attribute_Single doit être mis en œuvre si l'attribut de contrôle d'interface est mis en œuvre.

7.6.5.2 Réponse Get_Attribute_All

Pour les attributs de classe, étant donné qu'il n'existe qu'un seul attribut possible, la réponse Get_Attribute_All est identique à la réponse Get_Attribute_Single. Si aucun attribut n'est mis en œuvre, aucune donnée n'est renvoyée dans la partie correspondante de la réponse.

Pour les attributs d'instance, les attributs doivent être renvoyés dans l'ordre numérique jusqu'au dernier attribut mis en œuvre. Tous les zéros doivent être renvoyés pour les attributs Compteurs d'interface et Compteurs de support s'ils ne sont pas mis en œuvre mais que l'attribut de contrôle d'interface l'est.

7.6.6 Services spécifiques à la classe

7.6.6.1 Généralités

L'objet de liaison Ethernet doit prendre en charge les services spécifiques à la classe indiqués dans le Tableau 63.

Tableau 63 – Services spécifiques à la classe de l'objet de liaison Ethernet

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service
	Classe	Instance		
0x4C	N/A	Sous condition	Get_and_Clear	Permet d'obtenir, puis de supprimer, l'attribut spécifié (Compteurs d'interface ou Compteurs de support)

Le service Get_and_Clear doit uniquement être mis en œuvre si les attributs Compteurs d'interface et Compteurs de support le sont.

7.6.6.2 Service Get_and_Clear

Le service Get_and_Clear est un service spécifique à la classe. Il est uniquement pris en charge pour les attributs Compteurs d'interface et Compteurs de support. La réponse Get_and_Clear doit être identique à la réponse Get_Attribute_Single de l'attribut spécifié. Suite à la génération de la réponse, la valeur de l'attribut doit être nulle.

7.7 Objet DeviceNet

7.7.1 Présentation

L'objet DeviceNet doit fournir une interface cohérente de gestion de station aux couches de liaison physique et de données. Cet objet doit mettre les informations de diagnostic de ces couches à la disposition des applications client. Chaque nœud doit prendre en charge un objet DeviceNet par liaison.

NOTE Un dispositif peut contenir plusieurs nœuds.

7.7.2 Historique de révision

L'historique de révision de l'objet DeviceNet est décrit dans le Tableau 64.

Tableau 64 – Historique de révision de l'objet DeviceNet

Révision de classe	Description des modifications
01	Remise initiale
02	Remise pour la série CEI 61158

7.7.3 Attributs de classe

L'objet DeviceNet doit prendre en charge les attributs de classe spécifiés dans le Tableau 65.

Tableau 65 – Attributs de classe de l'objet DeviceNet

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x01	Requise	Get	Revision	UINT	Révision de cet objet	2 (révision 2)

7.7.4 Attributs d'instance

7.7.4.1 Généralités

L'objet DeviceNet doit prendre en charge les attributs d'instance spécifiés dans le Tableau 66.

Tableau 66 – Attributs d'instance de l'objet DeviceNet

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x01	Facultatif	Get/Set	MAC ID	USINT	Adresse de nœud	voir 7.7.4.2
0x02	Facultatif	Get/Set	Bit Rate	USINT	Vitesse en bits	voir 7.7.4.3
0x03	Facultatif	Get/Set	BOI	BOOL	Interruption de désactivation de bus	voir 7.7.4.4
0x04	Facultatif	Get/Set	Bus-Off Counter	USINT	Nombre de fois que CAN passe à l'état de désactivation de bus	voir 7.7.4.5
0x05	Facultatif	Get	Allocation Information	STRUCT de 2 octets		see 7.7.4.6
			Allocation choice	SWORD		voir 7.7.4.6.2
			Master's MAC ID	USINT	MAC ID du maître (de l'allocation)	voir 7.7.4.6.3
0x06	Conditionnel ^a	Get	MAC ID switch changed	BOOL	Le/Les commutateur(s) de l'adresse de nœud a/ont changé depuis la dernière mise sous tension / réinitialisation	0 = pas de modification 1 = modification depuis la dernière réinitialisation ou mise sous tension
0x07	Conditionnel ^b	Get	Bit rate switch changed	BOOL	Le/Les commutateur(s) de vitesse en bits a/ont changé depuis la dernière mise sous tension/réinitialisation	0 = pas de modification 1 = modification depuis la dernière réinitialisation ou mise sous tension
0x08	Conditionnel ^a	Get	MAC ID switch value	USINT	Valeur réelle du/des commutateur(s) d'adresse de nœud	0 à 99
0x09	Conditionnel ^b	Get	Bit rate switch value	USINT	Valeur réelle du/des commutateur(s) de vitesse en bits	0 à 9
0x0A	Facultatif ^c	Get/Set	Quick Connect	BOOL	Activation/Désactivation de la fonction de connexion rapide	0 = désactivation (valeur par défaut) 1 = activation
0x0B	Conditionnel ^d		Safety Network Number	SWORD[6]		Voir CEI 61784-3-2

ID attribut	Nécessité dans la mise en œuvre	Règle d'accès	Nom	Type de données	Description d'attribut	Sémantique des valeurs
0x0C	Facultatif	Get	Diagnostic counters	STRUCT de 34 octets	Liste des compteurs de diagnostic ISO 11898	Voir 7.7.4.7
			Diagnostic counters descriptor	WORD	Indique les compteurs de diagnostic pris en charge	
			Arbitration loss count	UINT	Voir ISO 11898	Plage = 0 à 65 535 Valeur par défaut = 0
			Overload count	UINT	Voir ISO 11898	
			Bit error count	UINT	Voir ISO 11898	
			Stuff error count	UINT	Voir ISO 11898	
			Ack error count	UINT	Voir ISO 11898	
			Form error count	UINT	Voir ISO 11898	
			CRC error count	UINT	Voir ISO 11898	
			Rx message loss count	UINT	Le sous-système ISO 11898 a détecté un message reçu perdu	
			Warning error count	UINT	Voir ISO 11898	Plage = 0 à 256
			Rx error counter	UINT	Recevoir compteur d'erreurs (voir ISO 11898)	
			Tx error counter	UINT	Transmettre compteur d'erreurs (voir ISO 11898)	Plage = 0 à 256
				UINT [5]	Réservé	Valeur par défaut = 0

^a Ces attributs sont obligatoires si une valeur différente du dispositif actuellement en ligne peut être attribuée au commutateur MAC ID.

^b Ces attributs sont obligatoires si une valeur différente du dispositif actuellement en ligne peut être attribuée au commutateur de vitesse en bits.

^c Cet attribut doit être stocké dans la mémoire non volatile.

^d Cet attribut est obligatoire pour les dispositifs de sécurité CPF 2. Les dispositifs non relatifs à la sécurité ne doivent pas mettre en œuvre cet attribut.

7.7.4.2 MAC ID

Cet attribut contient le MAC ID de ce dispositif. La plage de valeurs est comprise entre 0 et 63 décimales.

Un dispositif qui utilise un commutateur pour définir le MAC ID doit renvoyer une réponse d'erreur, avec le code d'erreur général 0x0E (l'attribut ne peut pas être défini), en réponse à une demande Set_Attribute_Single lorsque le paramètre de commutation est compris entre 0 et 63 et que le commutateur MAC ID est activé par d'autres paramètres. Si le paramètre de commutation MAC ID est supérieur à 63 (désactivé) ou que le commutateur MAC ID est désactivé par d'autres paramètres, le dispositif peut prendre en charge le paramètre de l'attribut MAC ID par la demande de service Set_Attribute_Single. Le dispositif doit fournir une indication visuelle (commutateur physique, DEL, par exemple) pour signaler que le commutateur MAC ID a été remplacé. Une panne récupérable mineure doit être déclarée lorsque le commutateur MAC ID est activé et indique un MAC ID valide, mais ne correspondant pas à l'adresse en ligne en cours du dispositif. Pendant la mise sous tension ou la réinitialisation, il convient qu'un dispositif passe à l'état de défaut de communication si

un paramètre non valide est attribué au commutateur MAC ID et que le paramètre de l'attribut MAC ID n'est pas pris en charge.

L'attribut MAC ID est considéré comme étant non volatil lorsque, une fois configuré, l'attribut doit être rappelé après un cycle de puissance ou la réinitialisation du dispositif. La valeur MAC ID 63 doit être attribuée par défaut à la configuration « prête à l'emploi ». Les scénarii suivants de détermination MAC ID s'appliquent uniquement aux dispositifs utilisant la mémoire non volatile pour stocker le MAC ID du dispositif.

1. Lorsque le commutateur MAC ID est valide et activé, sa valeur est chargée dans la mémoire non volatile lors de la mise sous tension ou de la réinitialisation du dispositif, et avant toute tentative de passage en ligne. La valeur du commutateur MAC ID n'est à aucun autre moment chargée dans la mémoire non volatile.
2. Les dispositifs doivent tenter de passer en ligne avec la valeur MAC ID stockée dans la mémoire non volatile ou, en cas de configuration « prête à l'emploi », avec une valeur MAC ID de 63.
3. Si un dispositif dont le commutateur MAC ID est désactivé reçoit une demande Set_Attribute_Single avec un MAC ID valide, le MAC ID non volatil doit être chargé avec le nouveau MAC ID.
4. Si un dispositif dont le commutateur MAC ID est activé reçoit une demande Set_Attribute_Single avec un MAC ID valide, le MAC ID non volatil ne doit pas changer et une réponse d'erreur portant le code d'erreur général 0x0E (l'attribut ne peut pas être défini) doit être renvoyée.

La modification de MAC ID demande au dispositif de supprimer tous les objets de connexion et d'exécuter de nouveau le diagramme d'état d'accès au réseau défini dans la CEI 62026-3.

7.7.4.3 Bit Rate

L'attribut Bit Rate indique la vitesse en bits sélectionnée. Les valeurs sont spécifiées dans le Tableau 67.

Tableau 67 – Valeurs de l'attribut Bit Rate

Valeur	Signification
00	125 kbit/s
01	250 kbit/s
02	500 kbit/s

Un dispositif qui utilise un commutateur pour définir la vitesse en bits doit renvoyer une réponse d'erreur, avec le code d'erreur général 0x0E (l'attribut ne peut pas être défini), en réponse à une demande Set_Attribute_Single lorsqu'une valeur valide est attribuée au commutateur de vitesse en bits, lequel n'est pas désactivé par d'autres paramètres. Si une valeur valide n'est pas attribuée au commutateur de vitesse en bits (désactivé) et/ou que le commutateur de vitesse en bits est désactivé par d'autres paramètres, le dispositif peut prendre en charge le paramètre de l'attribut Bit Rate par la demande de service Set_Attribute_Single. Le dispositif doit fournir une indication visuelle (commutateur physique, DEL, par exemple) pour signaler que le commutateur de vitesse en bits a été remplacé. Une panne récupérable mineure doit être déclarée lorsque le commutateur de vitesse en bits est activé et indique une vitesse en bits valide, mais ne correspondant pas à la vitesse en bits en ligne en cours du dispositif. Pendant la mise sous tension ou la réinitialisation, il convient qu'un dispositif passe à l'état de défaut de communication si un paramètre non valide est attribué au commutateur de vitesse en bits et que le paramètre de l'attribut Bit Rate n'est pas pris en charge.

L'attribut Bit Rate est considéré comme étant non volatil lorsque, une fois configuré, l'attribut doit être rappelé après un cycle de puissance ou la réinitialisation du dispositif. La

configuration « prête à l'emploi » doit être celle par défaut, de sorte que le dispositif soit en mesure de passer en ligne à 125 kbit/s. Les scénarii suivants de détermination de la vitesse en bits s'appliquent uniquement aux dispositifs utilisant une mémoire non volatile pour stocker la vitesse en bits du dispositif.

1. Lorsque le commutateur de vitesse en bits est valide et activé, sa valeur est chargée dans la mémoire non volatile lors de la mise sous tension ou de la réinitialisation du dispositif, et avant toute tentative de passage en ligne. La valeur du commutateur de vitesse en bits n'est à aucun autre moment chargée dans la mémoire non volatile.
2. Les dispositifs doivent tenter de passer en ligne avec la valeur de vitesse en bits stockée dans la mémoire non volatile ou, en cas de configuration « prête à l'emploi », être en mesure de passer en ligne à 125 kbit/s.
3. Si un dispositif dont le commutateur de vitesse en bits est désactivé reçoit une demande Set_Attribute_Single avec une vitesse en bits valide, l'attribut Bit Rate non volatil doit être chargé avec la nouvelle vitesse en bits.
4. Si un dispositif dont le commutateur de vitesse en bits est activé reçoit une demande Set_Attribute_Single avec une vitesse en bits valide, l'attribut Bit Rate non volatil ne doit pas changer et une réponse d'erreur portant le code d'erreur général 0x0E (l'attribut ne peut pas être défini) doit être renvoyée.

La modification de l'attribut Bit Rate ne prend pas effet tant que le dispositif n'est pas physiquement réinitialisé (un cycle de puissance ou un commutateur de réinitialisation, par exemple) ou réinitialisé le service Reset avec l'objet d'identité. Pendant cette période, la valeur de l'attribut Bit Rate ne correspond pas à la vitesse en bits réelle du réseau.

7.7.4.4 BOI (Bus-off interrupt)

L'attribut BOI est composé d'un bit qui définit la manière dont un dispositif CAN traite les interruptions de désactivation de bus. L'attribut BOI se trouve en position binaire 0 dans un octet pour les services Get_Attribute_Single/Set_Attribute_Single. Le reste des bits de l'octet doit être nul.

Tableau 68 – Valeurs de l'attribut BOI

Valeur	Signification
00	Maintient la puce CAN à son état de désactivation de bus (réinitialisation) en cas de détection d'une indication de désactivation de bus
01	Si possible, réinitialiser totalement la puce CAN et poursuivre la communication en cas de détection d'une indication de désactivation de bus

Si la valeur FALSE est affectée à l'attribut BOI (valeur nulle) et en cas de détection d'un événement de désactivation de bus de la puce CAN ISO 11898, suivre la procédure ci-dessous:

- la puce CAN est maintenue à l'état de réinitialisation/désactivation de bus;
- le dispositif passe à l'état de défaut de communication (voir CEI 62026-3).

Si la valeur TRUE est affectée à l'attribut BOI (définie sur un) et qu'un événement de désactivation de bus d'une puce CAN a été détecté, il peut être possible de renvoyer la puce CAN à son mode de fonctionnement normal et de poursuivre la communication en fonction du diagramme d'état d'accès au réseau défini dans la CEI 62026-3.

Si la valeur 1 est affectée à l'attribut BOI, le dispositif doit s'assurer qu'il ne se réinitialise pas indéfiniment et ne continue pas à produire des paquets corrompus sur le bus. Si ce n'est pas le cas, le dispositif interrompt toutes les communications sur le bus.

Les connexions ne sont pas nécessairement affectées lorsqu'un événement de désactivation de bus est détecté, et le dispositif peut poursuivre la communication. Les connexions préalablement établies peuvent être conservées ou supprimées (réinitialisation à chaud). Quoi qu'il en soit, l'algorithme de détection de MAC ID en double doit être de nouveau exécuté en fonction du diagramme d'état d'accès au réseau.

Comme indiqué dans le Tableau 66, la prise en charge de l'attribut BOI est facultative. S'il n'est pas pris en charge, un dispositif doit mettre en œuvre le comportement indiqué par la valeur d'attribut zéro (0) du Tableau 68.

7.7.4.5 Bus-off counter

L'attribut Bus-off Counter permet de compter le nombre de fois que la puce CAN est passé à l'état de déconnexion du bus (nombre d'interruptions de désactivation de bus). Le compteur comporte des valeurs comprises entre 0 et 255 décimales.

L'attribut Bus-off Counter est remis à zéro à la mise sous tension ou l'initialisation du dispositif.

L'attribut Bus-off Counter arrête le comptage lorsqu'il atteint le nombre maximal. Le compteur ne passe pas à zéro. Le compteur reste au nombre maximal tant qu'un service Set_Attribute_Single n'est pas exécuté.

L'objet DeviceNet remet l'attribut Bus-off Counter à zéro (0) à chaque fois qu'il reçoit une demande Set_Attribute_Single spécifiant l'attribut Bus-Off Counter. Les données Set_Attribute_Single ne sont pas utilisées et peuvent prendre n'importe quelle valeur. La transmission d'une demande Set_Attribute_Single à l'attribut Bus-off Counter contient tous les éléments permettant de réinitialiser le compteur.

7.7.4.6 Allocation Information

7.7.4.6.1 Présentation

L'attribut Allocation Information est destiné au jeu de connexions maître/esclave prédéfini. Son aide est requise si le jeu de connexions maître/esclave prédéfini est pris en charge. Il indique si le jeu défini dans la CEI 62026-3 a été alloué. S'il l'a été, cet attribut indique le dispositif qui a procédé à l'allocation et la/les connexion(s) actuellement allouées.

Cet attribut est modifié lorsqu'une réponse qui a abouti associée à un service Allocate_Master/Slave_Connection_Set (défini ultérieurement dans la présente section) est générée. Cet attribut ne peut pas être modifié par le service Set_Attribute_Single. Une réponse d'erreur dont le champ de code d'erreur général est défini sur 0x0E (l'attribut ne peut pas être défini) est renvoyée si une demande Set_Attribute_Single spécifie cet attribut.

L'attribut Allocation Information est composé des attributs Allocation choice et Master's MAC ID.

7.7.4.6.2 Allocation choice

L'attribut Allocation choice indique laquelle des connexions maître/esclave prédéfinies est activée (à l'état de configuration ou à l'état établi). Son format est spécifié dans la CEI 62026-3.

L'attribut Allocation choice est remis à 00 à la mise sous tension ou la réinitialisation.

7.7.4.6.3 Master's MAC ID

L'attribut Master's MAC ID contient le MAC ID du dispositif qui a alloué le jeu de connexions maître/esclave prédéfini par l'intermédiaire du service

Allocate_Master/Slave_Connection_Set. Il contient le champ Allocator's MAC ID copié à partir de la demande Allocate_Master/Slave_Connection_Set.

La plage de valeurs est comprise entre 0 et 63 et 255 décimales. Une valeur comprise entre 0 et 63 indique que le jeu de connexions maître/esclave prédéfini est actuellement alloué et signale le MAC ID du dispositif qui a exécuté l'allocation. La valeur 255 indique que le jeu de connexions maître/esclave prédéfini n'a pas été alloué.

L'attribut Master's MAC ID est initialisé à 255 (0xFF) à la mise sous tension/réinitialisation du dispositif.

7.7.4.7 Diagnostic counters

Cet attribut reporte les comptes des différents types d'erreurs réseau. Le premier champ (Diagnostic counters descriptor) identifie les compteurs pris en charge par le dispositif. Chaque bit indique si un compteur désigné est pris en charge. Si une valeur est attribuée, le compteur est pris en charge. Le Tableau 69 spécifie l'attribution de bit de compteur de diagnostic.

Tableau 69 – Description de bit des compteurs de diagnostic

Bit	Compteur	Peut être supprimé
0	Arbitration loss	Oui
1	Overload error	Oui
2	Bit error	Oui
3	Stuff error	Oui
4	Ack error	Oui
5	Form error	Oui
6	CRC error	Oui
7	Rx message loss	Oui
8	Warning error	Oui
9	Rx error	Non
10	Tx error	Non
11 – 15	Reservé (doit être nul 0)	N/A

Les champs restants indiquent les valeurs de compteur pour chaque compteur pris en charge. Les compteurs sont avancés de un pour chaque occurrence de l'erreur, sauf pour les compteurs Rx error et Tx error (champs 10 et 11), qui sont incrémentés et décrémentés par le périphérique CAN conformément à l'ISO 11898, et ne passe pas à zéro.

Tous les compteurs sont réinitialisés lorsque le dispositif entre l'envoi du message de demande de vérification (voir le diagramme de transition d'états d'accès à la liaison de la CEI 62026-3).

7.7.5 Services communs

7.7.5.1 Généralités

L'objet DeviceNet doit prendre en charge les services communs spécifiés dans le Tableau 70.

Tableau 70 – Services communs de l'objet DeviceNet

Code de service	Nécessité dans la mise en œuvre		Nom du service	Description du service
	Classe	Instance		
0x05	N/A	Conditionnel ^a	Reset	Utilisé pour réinitialiser cette instance de l'objet DeviceNet
0x10	N/A	Facultatif	Set_Attribute_Single	Définit les informations de configuration spécifiques au réseau ou au nœud
0x0E	Facultatif	Facultatif	Get_Attribute_Single	Obtient les informations de configuration spécifiques au réseau ou au nœud
^a Ce service est requis lorsque cette instance de l'objet DeviceNet peut être adressée par un mécanisme d'accès CPF 2 autre que l'interface de sous-réseau contrôlée par cette instance de l'objet DeviceNet.				

7.7.5.2 Reset

Lorsque l'objet DeviceNet reçoit une demande Reset, il

- détermine s'il peut fournir le type de réinitialisation demandée;
- répond à la demande;
- exécute le type de réinitialisation demandée.

Le service commun Reset contient le paramètre spécifique à l'objet spécifié dans le Tableau 71.

Tableau 71 – Paramètre du service Reset

Nom	Type	Description des paramètres de demande	Sémantique des valeurs
Type	USINT	Type de réinitialisation	Voir Tableau 72

Le paramètre type du service commun Reset contient les spécifications d'énumération indiquées dans le Tableau 72.

Tableau 72 – Valeurs de paramètre du service Reset

Valeur	Type de réinitialisation
0	Émuler de la meilleure façon possible la puissance de cyclage sur la liaison de réseau que l'instance de l'objet DeviceNet représente. La valeur est celle par défaut si ce paramètre est ignoré
1	Renvoie de la meilleure façon possible à la configuration prête à l'emploi, puis émule la puissance de cyclage sur le réseau aussi proche que possible
2	Renvoie de la meilleure façon possible à la configuration prête à l'emploi, puis émule la puissance de cyclage sur le réseau aussi proche que possible Le MAC ID et la vitesse en bits ne changent pas suite à cette réinitialisation
3 – 99	Réservé
100– 199	Comportement de réinitialisation spécifique au fournisseur
200 – 255	Réservé

7.7.6 Services spécifiques à la classe

7.7.6.1 Généralités

L'objet DeviceNet doit prendre en charge les services spécifiques à la classe spécifiés dans le Tableau 73.

Tableau 73 – Services spécifiques à la classe de l'objet DeviceNet

Code de service	Nécessité dans la mise en œuvre	Nom du service	Description du service
0x4B	Facultatif	Allocate_Master/Slave_Connection_Set	Demande l'utilisation du jeu de connexions maître/esclave prédéfini
0x4C	Conditionnel ^a	Release_Master/Slave_Connection_Set	Indique que les connexions spécifiées dans le jeu de connexions maître/esclave prédéfini ne sont plus souhaitées. Ces connexions doivent être libérées (supprimées)
0x4D	Conditionnel ^b	Clear_Diagnostics	Supprime les compteurs de diagnostic renvoyés dans l'attribut 12
^a Le service Release_Master/Slave_Connection_Set est requis si le service Allocate_Master/Slave_Connection_Set est mis en œuvre dans le dispositif. ^b Ce service doit être pris en charge si les compteurs qui peuvent être supprimés sont mis en œuvre dans l'attribut Diagnostic Counters.			

7.7.6.2 Allocate_Master/Slave_Connection_Set, Release_Master/Slave_Connection_Set

Ces services permettent d'allouer et de désallouer le jeu de connexions maître/esclave prédéfini décrit dans la CEI 62026-3.

Un dispositif qui se comporte comme le client dans le jeu de connexions maître/esclave prédéfini est appelé maître. Un dispositif qui se comporte comme le serveur dans le jeu de connexions maître/esclave prédéfini est appelé esclave. Dans les limites des descriptions du service, un maître et/ou esclave se présente comme une unité fonctionnelle dans un dispositif de communication.

Un dispositif qui souhaite fonctionner comme le maître d'un autre dispositif doit en premier lieu allouer le jeu de connexions maître/esclave prédéfini à l'intérieur de l'esclave. Le jeu de connexions maître/esclave prédéfini peut être alloué à un seul maître à un instant donné. L'ensemble du jeu de connexions est alloué, et le maître utilise un sous-ensemble sélectionné de connexions issues de l'ensemble (c'est-à-dire Bit Strobe uniquement ou Poll uniquement). Pour « abandonner » son esclave, le maître libère toutes les connexions, l'esclave « désallouant » le jeu de connexions maître/esclave prédéfini.

7.7.6.3 Clear_Diagnostics

Ce service supprime (remet à zéro) les compteurs qui peuvent l'être dans l'attribut 12 (compteurs de diagnostic). Les compteurs qui peuvent être supprimés sont indiqués dans le Tableau 69.

7.8 Objet de configuration de connexion (CCO)

7.8.1 Présentation

Cet objet définit une interface de création, de configuration et de contrôle des connexions d'un dispositif. Cette spécification ne définit ni ne contraint le fonctionnement du diagramme d'état de gestion des connexions du dispositif. Les attributs de classe utilisés par cet objet sont présentés dans le Tableau 75.

7.8.2 Historique de révision

L'historique de révision de l'objet de configuration de connexion est décrit dans le Tableau 74.