IEC 61158-4-2

Edition 5.0 2023-03

INTERNATIONAL STANDARD

colour inside

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

IEC 61158-4-2:2023-03(en)

IEC Secretariat
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
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 corrigendum or an amendment might have been published.

IEC publications search - webstore.iec.ch/advsearchform
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 It also gives information on projects, replaced and withdrawn publications.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 once a month by email.

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: sales@iec.ch.

IEC Products & Services Portal - products.iec.ch
Discover our powerful search engine and read freely all the publications previews. With a subscription you will always have access to up to date content tailored to your needs.

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 300 terminological entries in English and French, with equivalent terms in 19 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC 61158-4-2

Edition 5.0 2023-03

INTERNATIONAL STANDARD

colour inside

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

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ICS 25.040.40; 35.100.20; 35.110

ISBN 978-2-8322-6554-3

Warning! Make sure that you obtained this publication from an authorized distributor.

CONTENTS

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 itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

Attention is drawn to the fact that the use of the associated protocol type is restricted by its 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 layer protocol type to be used with other layer protocols of the same type, or in other type combinations explicitly authorized by its intellectual-property-right holders.

NOTE Combinations of protocol types are specified in the IEC 61784-1 series and the IEC 61784-2 series.

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

This fifth edition cancels and replaces the fourth edition published in 2019. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

a) update of normative and bibliographic references;
b) use of more inclusive terminology ("master" and "slave" replaced, mainly in 7.3 and 7.7;
c) new STIME, UTIME, NTIME, STRINGI and EPATH data types in 6.1.3;
d) updates, addition of diagnostics connection points and new service for TCP/IP interface object in 7.5;
e) addition of diagnostics connection points and new service for Ethernet Link object in 7.6;
f) update of Get/Set_Attributes_All parameters for the Connection Configuration object in 7.8;
g) addition of diagnostics connection points and new service for DLR object in 7.9;
h) extensions and clarifications of Port object in 7.11;
i) addition of diagnostics connection points and new service for PRP/HSR Protocol object in 7.12;
j) addition of LLDP Management and LLDP Data Table objects in 7.1, 7.14 and 7.15;
k) addition of LLDP protocol support in Clause 12;
l) addition of a combined module/network indicator in A.2.4.5;
m) removal of all references to CPF and CPs (material moved to profile documents);
n) miscellaneous editorial corrections.

The text of this International Standard is based on the following documents:

Draft	Report on voting
65C/1202/FDIS	65C/1243/RVD

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

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/publications.

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

The committee has decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

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

IMPORTANT – The "colour inside" logo on the cover page of this document indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INTRODUCTION

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

The data-link protocol provides the data-link service by making use of the services available from the physical layer. The primary aim of this document 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 implementers 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 document is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this document together with other standards positioned within the OSI or fieldbus reference models, otherwise incompatible systems could work together in any combination.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of patents. 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 is registered with IEC. Information may be obtained from the patent database available at http://patents.iec.ch.

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

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 part of IEC 61158 specifies a main protocol with the following characteristics.

- 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 can 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 µs.
- 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.

This document specifies additional lower layers protocols or implementations of additional lower layers protocols for use in combination with ISO/IEC/IEEE 8802-3.

This document specifies a set of corresponding objects providing a consistent management interface to the lower layers.

1.2 Specifications

This document 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 document, 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 capabilities of an implementation, and thus its applicability to various time-critical communications needs.

1.5 Conformance

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

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

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

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

IEC 61158-3-2:2023, *Industrial communication networks – Fieldbus specifications – Part 3-2: Data-link layer service definition – Type 2 elements*

IEC 61158-5-2:2023, *Industrial communication networks – Fieldbus specifications – Part 5-2: Application layer service definition – Type 2 elements*

IEC 61158-6-2:2023, *Industrial communication networks – Fieldbus specifications – Part 6-2: Application layer protocol specification – Type 2 elements*

IEC 61588, *Precision clock synchronization protocol for networked measurement and control systems*

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

IEC 62026-3:2014, *Low-voltage switchgear and controlgear – Controller-device interfaces (CDIs) – Part 3: DeviceNet*

IEC 62439-3:2016[1], *Industrial communication networks – High availability automation networks – Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR)*

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

ISO/IEC 13239, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*

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/IEEE 8802-3, *Telecommunications and exchange between information technology systems − Requirements for local and metropolitan area networks – Part 3: Standard for Ethernet*

ISO 11898-1:2015, *Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*

IEEE Std 802.1AB-2016, *IEEE Standard for local and metropolitan area networks – Station and Media Access Control Connectivity Discovery*

IEEE Std 802.1ABcu-2021, *Standard for local and metropolitan area networks – Station and Media Access Control Connectivity Discovery Amendment: YANG Data Model*

IEEE Std 802.1Q-2018, *IEEE standard for local and metropolitan area networks – Bridges and bridged networks*

IEEE Std 802.3-2018, *IEEE Standard for Ethernet*

IETF RFC 951, W.J. Croft, J. Gilmore, *Bootstrap Protocol*, September 1985, available at https://www.rfc-editor.org/info/rfc951 [viewed 2022-02-18]

IETF RFC 1213, K. McCloghrie, M. Rose, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, March 1991, available at https://www.rfc-editor.org/info/rfc1213 [viewed 2022-02-18]

IETF RFC 1542, W. Wimer, *Clarifications and Extensions for the Bootstrap Protocol*, October 1993, available at https://www.rfc-editor.org/info/rfc1542 [viewed 2022-02-18]

IETF RFC 1643, F. Kastenholz, *Definitions of Managed Objects for the Ethernet-like Interface Types*, July 1994, available at https://www.rfc-editor.org/info/rfc1643 [viewed 2022-02-18]

IETF RFC 2131, R. Droms, *Dynamic Host Configuration Protocol*, March 1997, available at https://www.rfc-editor.org/info/rfc2131 [viewed 2022-02-18]

IETF RFC 2132, S. Alexander, R. Droms, *DHCP Options and BOOTP Vendor Extensions*, March 1997, available at https://www.rfc-editor.org/info/rfc2132 [viewed 2022-02-18]

IETF RFC 2863, K. McCloghrie, F. Kastenholz, *The Interfaces Group MIB*, June 2000, available at https://www.rfc-editor.org/info/rfc2863 [viewed 2022-02-18]

IETF RFC 3418, R. Presuhn, *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)*, December 2002, available at https://www.rfc-editor.org/info/rfc3418 [viewed 2022-02-18]

IETF RFC 3635, J. Flick, *Definitions of Managed Objects for the Ethernet-like Interface Types*, September 2003, available at https://www.rfc-editor.org/info/rfc3635 [viewed 2022-02-18]

IETF RFC 4541, M. Christensen, K. Kimball, F. Solensky, *Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches*, May 2006, available at https://www.rfc-editor.org/info/rfc4541 [viewed 2022-02-18]

IETF RFC 5227:2008, S. Cheshire, *IPv4 Address Conflict Detection*, July 2008, available at https://www.rfc-editor.org/info/rfc5227 [viewed 2022-02-18]

3 Terms, definitions, symbols, abbreviated terms and conventions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at https://www.electropedia.org/
- ISO Online browsing platform: available at https://www.iso.org/obp

3.1 Reference model terms and definitions

This document 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]
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 document 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

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

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

3.3.2
DLSAP
distinctive point at which DL-services are provided by a single DL-entity to a single higher-layer entity

Note 1 to entry: Definition is provided in a generic manner "(N)-SAP" in ISO/IEC 7498-1.

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 1 to entry: 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 1 to entry: 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 1 to entry: 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

Note 1 to entry: 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 1 to entry: 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 1 to entry: 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 can 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

Note 1 to entry: Its description includes the relationship between attribute values and services.

3.4.7
bit
unit of information consisting of a 1 or a 0

Note 1 to entry: 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) object which uses the services of another (server) object to perform a task
2) 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 1 to entry: 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 cannot hear the moderator DLPDU but can hear other link traffic

3.4.16
device
physical hardware connection to the link

Note 1 to entry: A device may contain more than one node.

3.4.17
DLPDU
data-link protocol data unit

Note 1 to entry: 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 1 to entry: Identification of the target node(s) is included in the two octet tag.

3.4.23
frame
denigrated synonym for DLPDU

3.4.24
Frame Check Sequence
FCS
redundant data derived from a block of data within a DLPDU (frame), using a hash function, and stored or transmitted together with the block of data, in order to detect data corruption

3.4.25
generic tag
three octet identifier (tag) which identifies a specific piece of application information

3.4.26
guardband
time slot allocated for the transmission of the moderator DLPDU

3.4.27
implicit token
mechanism that governs the right to transmit

Note 1 to entry: 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.

3.4.28
implicit token register
register that contains the MAC ID of the node that holds the right to transmit

3.4.29
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 1 to entry: The terms object, instance, and object instance are used to refer to a specific instance.

3.4.30
instance attribute
attribute whose value is unique to an object instance and whose definition is shared by all instances of an object

3.4.31
instantiated
object that has been created in a device

3.4.32
Keeper
object responsible for distributing link configuration data to all nodes on the link

3.4.33
link
collection of nodes with unique MAC IDs, attached to Ph-segments connected by Ph-repeaters

3.4.34
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

3.4.35
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

3.4.36
M_symbol(s)
symbols that represent the data bits and related information encoded and transmitted by the PhL

3.4.37
maximum scheduled node
node with highest MAC ID that can use scheduled time on a link

3.4.38
maximum unscheduled node
node with highest MAC ID that can use unscheduled time on a link

3.4.39
Message Router
object within a node that distributes messaging requests to the appropriate application objects

3.4.40
moderator
node with the lowest MAC ID that is responsible for transmitting the moderator DLPDU

3.4.41
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

3.4.42
multipoint connection
connection from one node to many

Note 1 to entry: Multipoint connections allow messages from a single producer (publisher) to be received by many consumer (subscriber) nodes.

3.4.43
network
series of nodes connected by some type of communication medium

Note 1 to entry: The connection paths between any pair of nodes can include repeaters, routers and gateways.

3.4.44
network access monitor
NAM
component within a node that manages communication with a temporary node connected to the nodes local NAP

3.4.45
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

3.4.46
network address or node address
node's address on the link

Note 1 to entry: This is also called MAC ID.

3.4.47
network status indicators
indicators on a node indicating the status of the Physical and Data Link Layers

3.4.48
network update time
NUT
repetitive time interval in which data can be sent on the link

3.4.49
node
logical connection to a local link, requiring a single MAC ID

Note 1 to entry: A single physical device can appear as many nodes on the same local link. For the purposes of this protocol, each node is considered to be a separate DLE.

3.4.50
non-concurrence
situation where a transmission is received from an unexpected MAC ID, which appears to violate the time based access protocol

Note 1 to entry: This can occur when a connection is made between two working links that are not synchronized with each other but who have the same configuration information.

3.4.51
non-data symbol
PhL symbol which violates the requirements of Manchester biphase L encoding

3.4.52
object
abstract representation of a particular component within a device

Note 1 to entry: 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.

3.4.53
object specific service
service defined by a particular object class to perform a required function which is not performed by a common service

Note 1 to entry: An object specific service is unique to the object class which defines it.

3.4.54
originator
client responsible for establishing a connection path to the target

3.4.55
permanent node
node whose connection to the network does not utilize the network access port (NAP) PhL variant

Note 1 to entry: This node may optionally support a NAP PhL variant to allow temporary nodes to connect to the network.

3.4.56
produce
act of sending data to be received by a consumer

3.4.57
producer
node that is responsible for sending data

3.4.58
redundant media
system using more than one medium to help prevent communication failures

3.4.59
requested packet interval
RPI
measure of how frequently the originating application requires the transmission of Lpackets or data packets from the target application

3.4.60
rogue
node that has received a moderator DLPDU that disagrees with the link configuration currently used by this node

3.4.61
scheduled
data transfers that occur in a deterministic and repeatable manner on predefined NUTs

3.4.62
server
object which provides services to another (client) object

3.4.63
service
operation or function than an object and/or object class performs upon request from another object and/or object class

Note 1 to entry: 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.

3.4.64
slot time
maximum time required for detecting an expected transmission

Note 1 to entry: Each node waits a slot time for each missing node during the implied token pass.

3.4.65
start delimiter
unique set of M_symbols that identifies the beginning of a DLPDU

3.4.66
supernode
DLE with a MAC ID of zero

Note 1 to entry: This DLE DL-address is reserved for special DLL functions

3.4.67
table unique identifier
TUI
unique reference code used by the ControlNet object and the Keeper object to reference a consistent set of link configuration attribute

3.4.68
tag
shorthand name for a specific piece of application information, two or three octets in length

3.4.69
target
end-node to which a connection is established

3.4.70
temporary node
same as transient node

3.4.71
tMinus
number of NUTs before a new set of link configuration parameters are to be used

3.4.72
tone
instant of time which marks the boundary between two NUTs

3.4.73
tool
executable software program which interacts with the user to perform some function

EXAMPLE Link scheduling software (LSS).

3.4.74
transaction id
field within a UCMM header that matches a response with the associated request, which a server echoes in the response message

3.4.75
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

3.4.76
Unconnected Message Manager
UCMM
component within a node that transmits and receives unconnected explicit messages and sends them directly to the Message Router object

3.4.77
unconnected service
messaging service which does not rely on the set up of a connection between devices before allowing information exchanges

3.4.78
unscheduled
data transfers that use the remaining allocated time in the NUT after the scheduled transfers have been completed

3.4.79
vendor ID
identification of each product manufacturer/vendor by a unique number

Note 1 to entry: Vendor IDs are assigned by the ODVA, Inc. organization (see <www.odva.org>).

3.5 Type 2 symbols and abbreviated terms

ACD	Address Conflict Detection	
ACM	Access control machine	
BOOTP	Bootstrap Protocol	[IETF RFC 951]
CA	Clock accuracy	
CAN	Controller Area Network	[ISO 11898-1]
CCITT	Consultative Committee for International Telephony and Telegraphy	
COP	Connection originator password	
DANP	Doubly Attached Node operating PRP	
DHCP	Dynamic Host Configuration Protocol	[IETF RFC 2131]
DLR	Device level ring	
DSCP	DiffServ code point	
FCS	Frame Check Sequence	
HSR	High-availability Seamless Redundancy	[IEC 62439-3]
IP	Internet Protocol	[IETF RFC 791]
LAN	Local Area Network	
LED	Light emitting diode	
LRE	Link Redundancy Entity (associated with PRP)	
LSS	Link scheduling software	
MAC ID	MAC address of a node	
MSTP	Multiple spanning tree protocol	[IEEE Std 802.1Q-2018]
NAM	Network attachment monitor	
NAP	Network access port	
ND	Non-data symbol	
PRP	Parallel Redundancy Protocol	[IEC 62439-3]
NUT	Network update time	
PT	Programming terminal (a temporary network connection)	
PTP	Precision Time Protocol	[IEC 61588]
QoS	Quality of service	
Rcv	Receive	
RedBox	PRP Redundancy Box (for attaching non-PRP nodes)	
RPI	Requested Lpacket interval	
RSTP	Rapid spanning tree protocol	[IEEE Std 802.1Q-2018]
Rx	Receive	

RxLLC	Receive logical link control	
RxM	Receive machine	
SAN	Singly Attached Node (in a PRP topology)	
SEM	State event matrix	
SMAX	MAC ID of the maximum scheduled node	
STD	State transition diagram, used to describe object behavior	
STP	Spanning tree protocol	[IEEE Std 802.1Q-2018]
TCP	Terminal Control Protocol	[IETF RFC 793]
TFTP	Trivial File Transfer Protocol	[IETF RFC 1350]
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	
VLAN	Virtual LAN	

3.6 Conventions for station management objects

Objects for station management are specified in Clause 7, using attributes, services and behavior.

Object attributes are specified in tables formatted according to Table 1.

Table 1 – Format of attribute tables

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
See a)	See b)	See c)	See d)	See e)	See 6.1.3	See f)	See g)

a) "Attribute ID" is the identification value assigned to an attribute.

b) "Need in implementation" specifies whether or not the attribute is necessary in the implementation. An attribute may be optional, required or conditional.

A conditional attribute is required if certain object behaviors and/or attributes are implemented as defined by the class.

If an Attribute is optional, then a default value shall be defined in the semantics. An optional attribute's default value shall provide the same behavior as if the attribute was not implemented. If the default value of an optional attribute does not match the behavior of the object when the object does not implement the attribute, the object definition shall declare the behavior.

In all cases, the term "default" indicates a "factory default" as shipped from the vendor.

c) "Access rule" specifies how a requestor can access an attribute. The definitions for access rules are:

- Settable (Set) – The attribute shall be accessed by at least one of the set services. For enumerated attributes, if the device supports only one of the values and the object definition does not require more than one value to be supported, then the attribute may be implemented as Get only.

Settable attributes, unless otherwise specified by the object definition, shall be also accessed by get services.

- Gettable (Get) – The attribute shall be accessed by at least one of the get services.

d) "NV" indicates whether an attribute value is maintained through power cycles. This column is used in object definitions where non-volatile storage of attribute values is required. An entry of "NV" indicates value shall be saved, "V" means not saved.

e) Name refers to the attribute.

f) Description of Attribute provides general information about the attribute.

g) Semantics of Values specifies the meaning of the value of the attribute.

4 Overview of the data-link 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 2:

Table 2 – 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 1. The arrowheads illustrate the primary direction of flow of data and control.

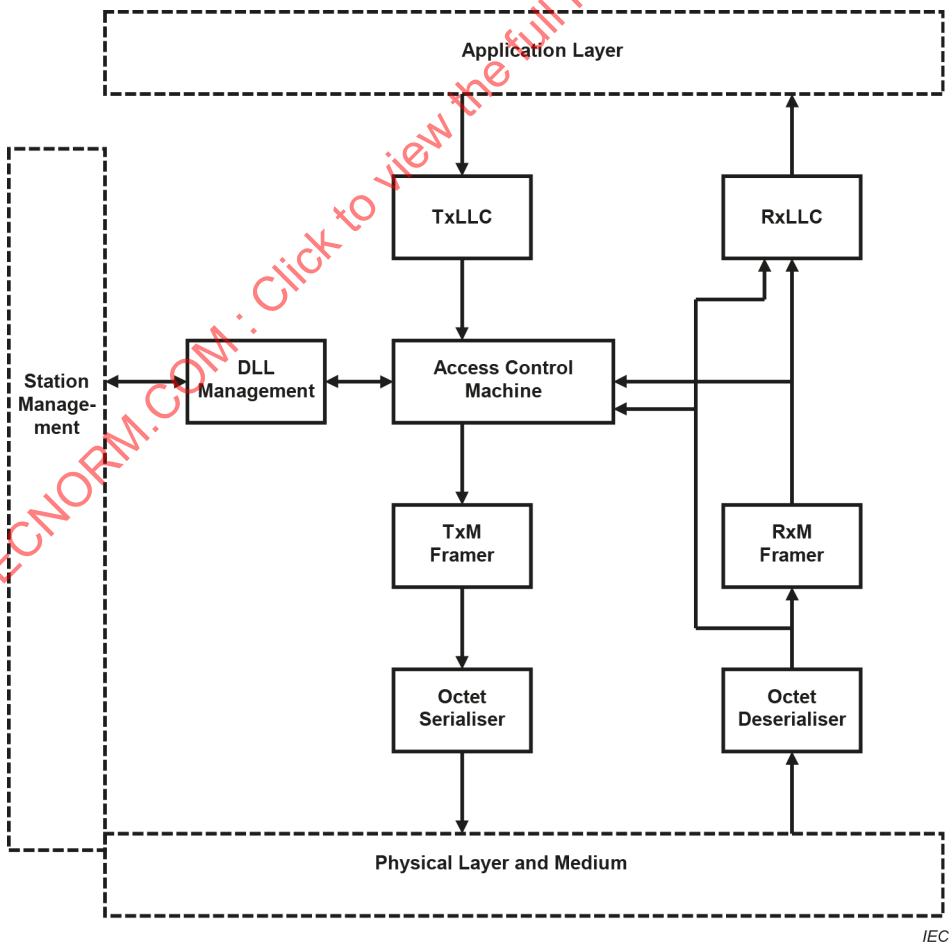


Figure 1 – Data-link layer internal architecture

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 is regulated at two levels,

a) 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,

b) to provide a democratic distribution of unscheduled opportunities for arbitrary communications, generally in a cyclic but asynchronous manner.

Accurate schedule timing as provided by a) 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.

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

   NOTE 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 the 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 active 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.

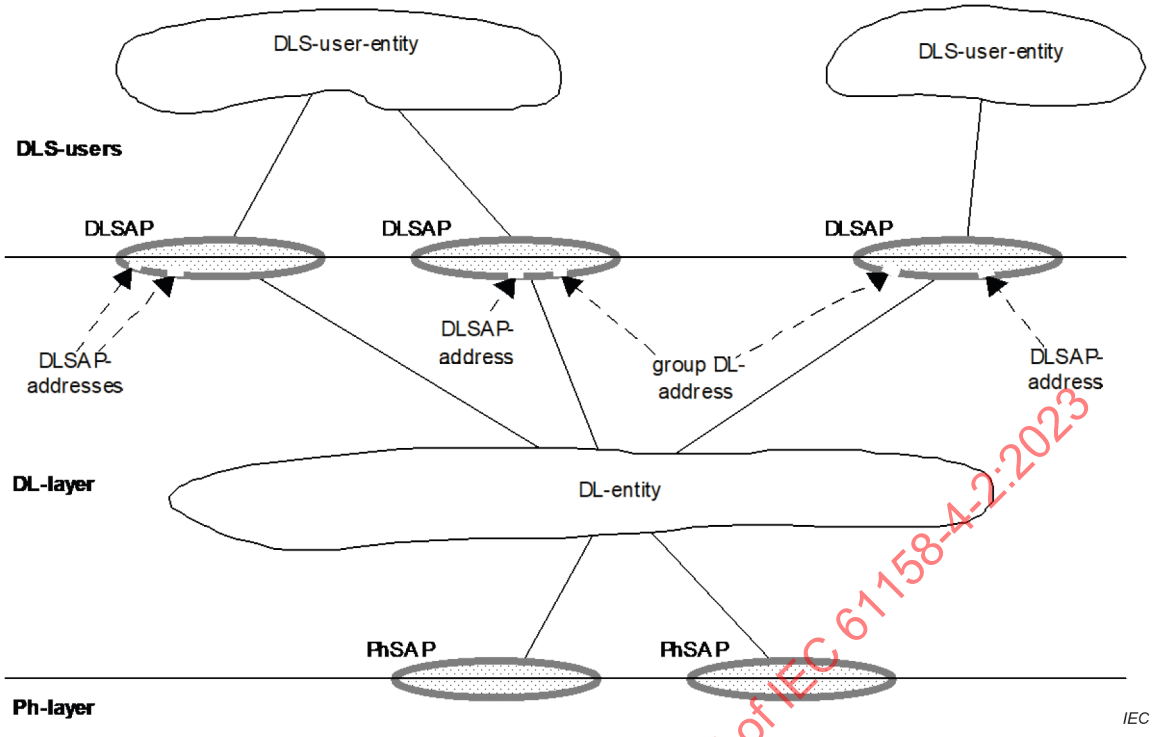
NOTE High and Low priority are used only in a local sense to set the order of servicing among locally submitted, unscheduled DLS-user-data.

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).

NOTE The definition of DLSAP, derived from ISO/IEC 7498-1, is explained here to facilitate understanding of the critical distinction between DLSAPs and their DL-addresses (see Figure 2).



NOTE 1 DLSAPs and physical layer service access points (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.

Figure 2 – Relationships of DLSAPs, DLSAP-addresses, and group DL-addresses

A single DLE may have multiple DLSAP-addresses and group DL-addresses associated with a single DLSAP.

Subclause 4.3 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 (see 4.3.2, 4.3.3 and 4.3.4).

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 to 99, values from 100 to 254 are reserved.

MAC ID
1 octet

Figure 3 – Basic structure of a MAC ID address

Table 3 specifies the use of MAC ID addresses.

Table 3 – 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.

Generic tag
3 octets

Figure 4 – Basic structure of a generic tag address

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

NOTE 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 4. 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 4 – Fixed tag service definitions

Fixed tag service	Meaning
0x00	moderator
0x01 – 0x08	vendor specific
0x09	ping request
0x0A – 0x14	vendor specific
0x15	tMinus
0x16 – 0x28	vendor specific

Fixed tag service	Meaning
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

Subclauses 4.4.2 to 4.4.3 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 5), 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 5. The M_0 and M_1 will be encoded into Ph_Symbols that represent Manchester biphase 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 5.

Table 5 – 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

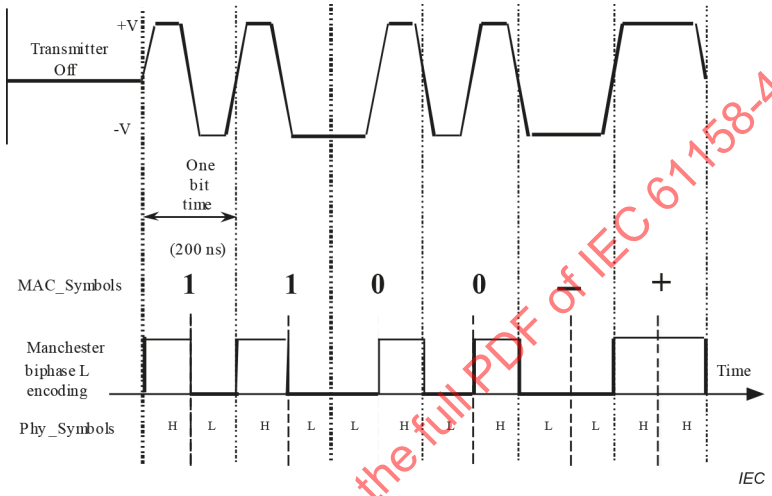


Figure 6 – M_symbols and Manchester encoding at 5 MHz

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 4.4.3.

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 5. Valid M_symbols shall be M_0 {0}, M_1 {1}, M_ND+ {+}, and M_ND– {-} as shown in Table 6.

Table 6 – 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 7. 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 7 – Truth table for ph_status_indication

Ph_status_indication	Ph_frame_indication	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 6. 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 can 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.

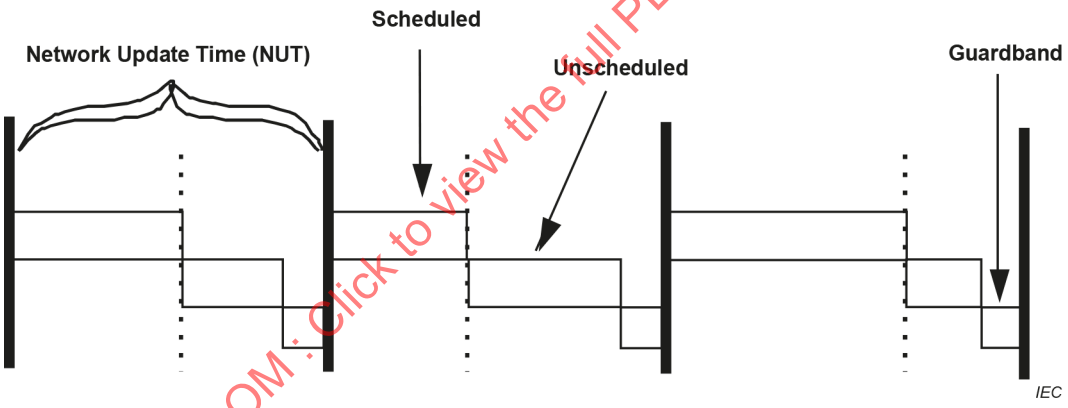


Figure 7 – NUT structure

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.

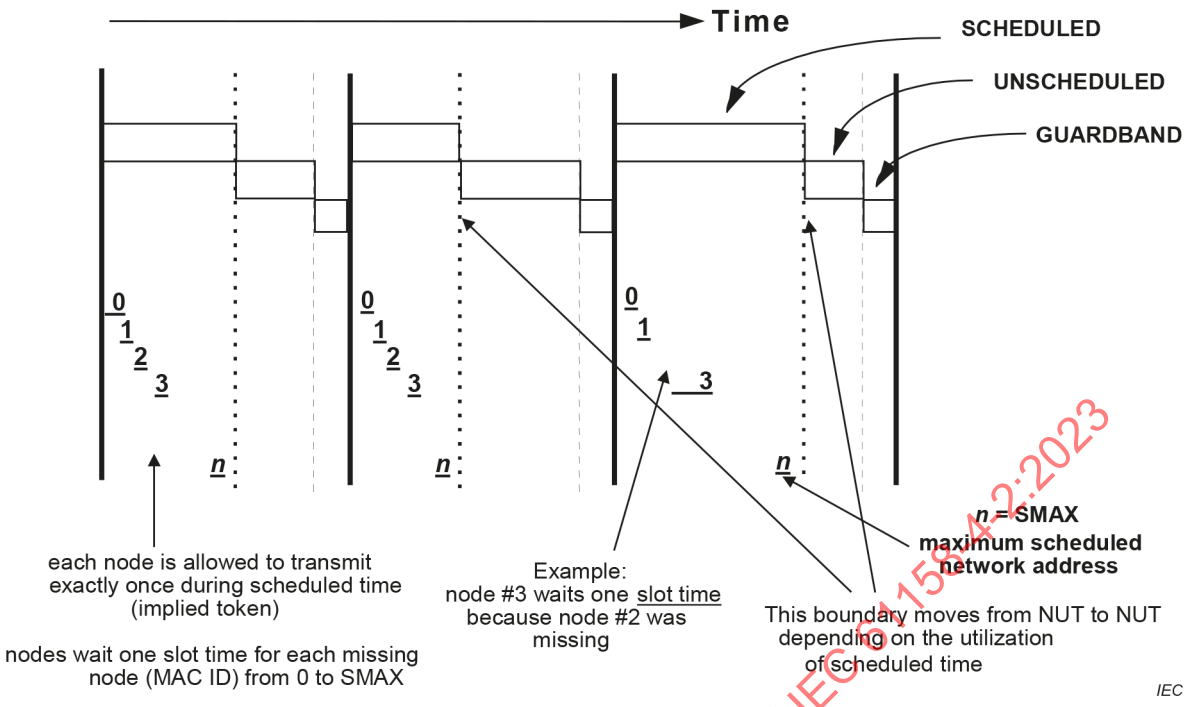


Figure 8 – Media access during scheduled time

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 can 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.

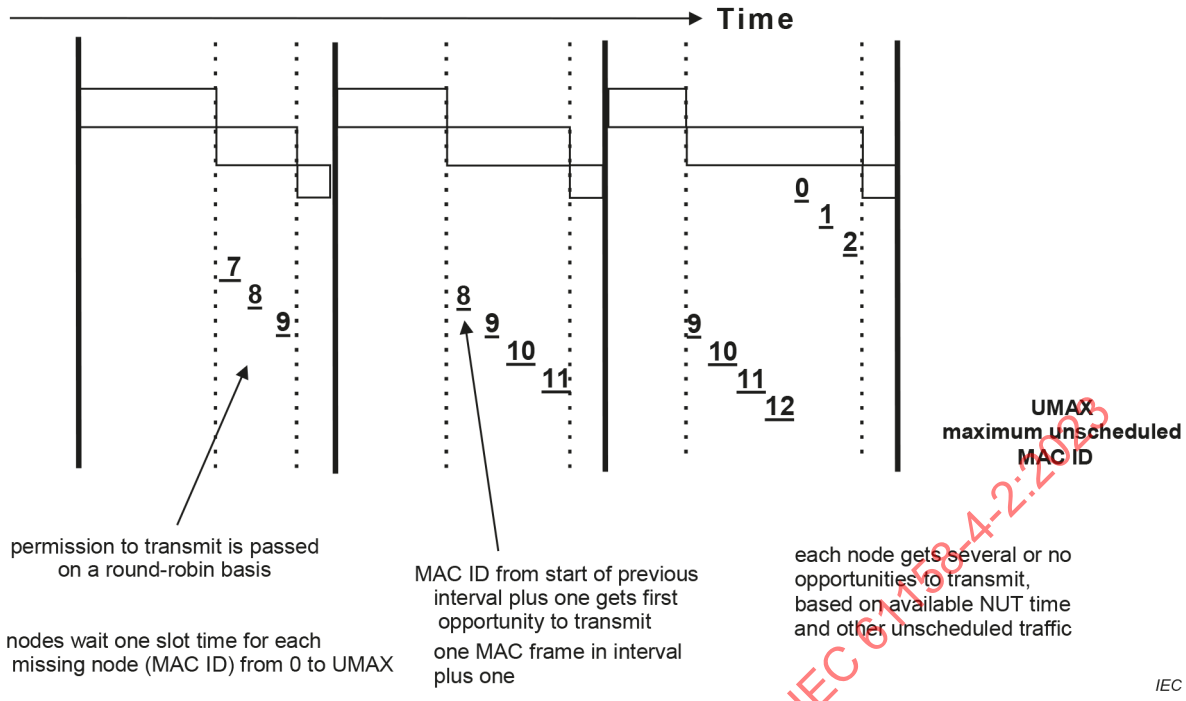


Figure 9 – Media access during unscheduled time

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 can 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 can 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 1.

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.

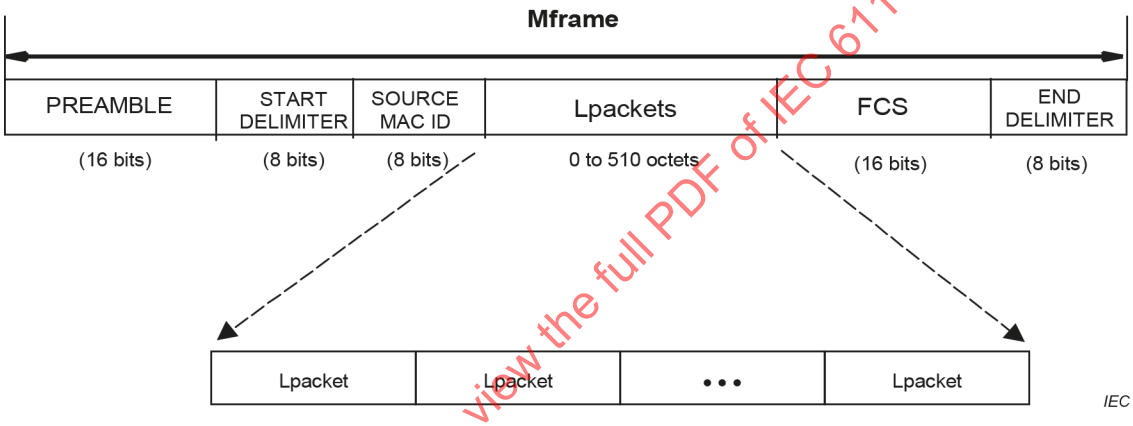


Figure 10 – DLPDU format

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 5.3.8.2, 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 document, as in other international standards (for example, ISO/IEC 13239, ISO/IEC/IEEE 8802-3 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" (see for example *Error Correcting Codes* from W.W.Peterson and E.J.Weldon,Jr.) 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 document are specified in Table 8.

Table 8 – 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 ≤ 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 13239 (HDLC).

NOTE 3 The remainder R(x) is 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 $m_1 \ldots m_k$ and the FCS bits are $f_{n-k-1} \ldots f_0$, where

$m_1 \ldots m_8$	form the first octet sent,
$m_{8N-7} \ldots m_{8N}$	form the *N*th octet sent,
$f_7 \ldots f_0$	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

$$M(X) = m_1X^{k-1} + m_2X^{k-2} + \ldots + m_{k-1}X^1 + m_k \quad (1)$$

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

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

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

$$\begin{aligned} D(X) &= M(X)\,X^{n-k} + F(X) \\ &= m_1X^{n-1} + m_2X^{n-2} + \ldots + m_kX^{n-k} + f_{n-k-1}X^{n-k-1} + \ldots + f_0 \\ &= m_1X^{n-1} + m_2X^{n-2} + \ldots + m_kX^{16} + f_{15}X^{15} + \ldots + f_0 \quad \text{(for the case of k = 15)} \end{aligned} \quad (3)$$

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

The redundant check bits $f_{n-k-1} \ldots f_0$ 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

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

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

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

That is,

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

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

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

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

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***

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

NOTE 1 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)

$$R(X) = L(X)\,X^{u} + V(X)\,X^{n-k} \ \text{(modulo } G(X))$$
$$= r_{n-k-1} X^{n-k-1} + \ldots + r_0 \tag{9}$$

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.

$$E(X) = D(X) + V(X) \tag{10}$$

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 2 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.

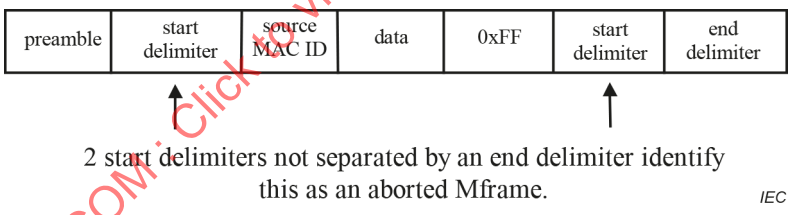


Figure 11 – Aborting a DLPDU during transmission

NOTE 1 Some implementations of this specification can 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.

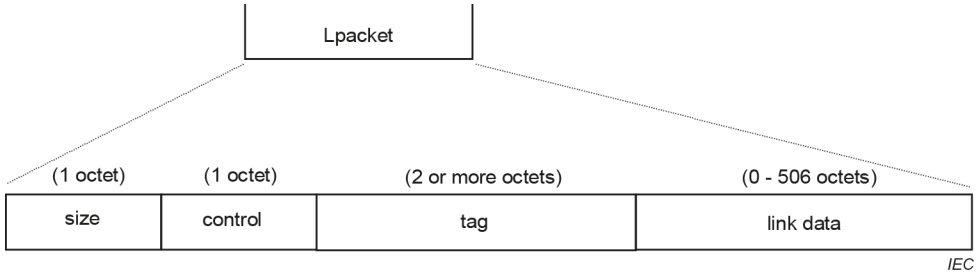


Figure 12 – Lpacket format

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.

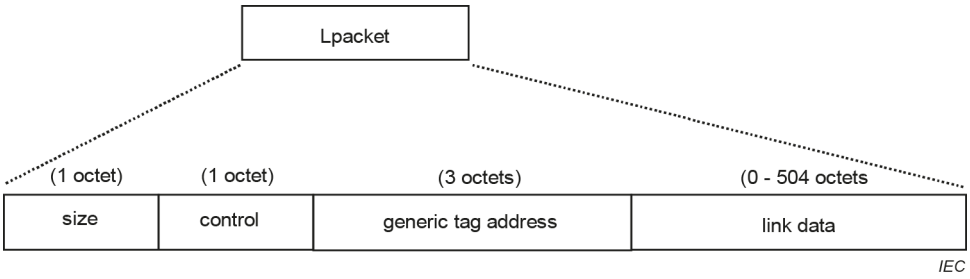


Figure 13 – Generic tag Lpacket format

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 destination node and the address of the destination node.

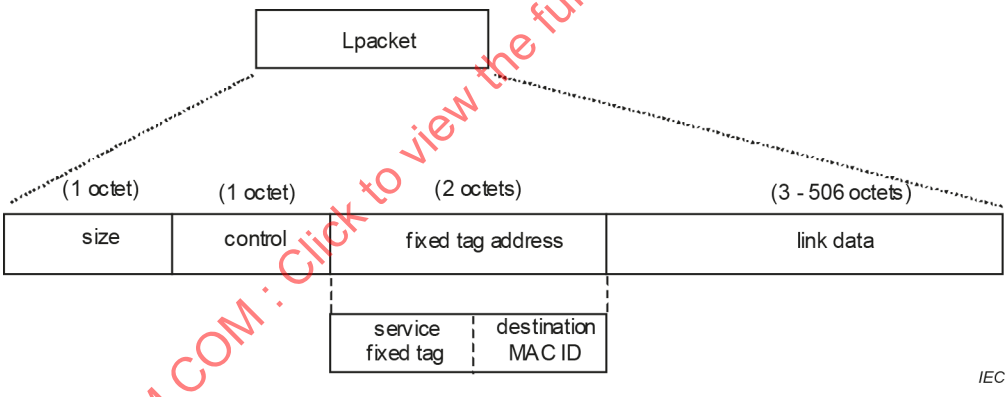


Figure 14 – Fixed tag Lpacket format

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 Framer (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 9. 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 9 – 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

DLL item	Description	Location
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
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 active Keeper and backup Keeper roles when active 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
Connection Configuration object	provides an interface to create, configure and control connections in a device	see 7.8
Port object	describes the communication interfaces that are present on the device and visible to Type 2 networks	see 7.11

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 could 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 10. Additional detail is specified in IEC 61158-6-2.

Table 10 – 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		
STIME	System time (nanoseconds)		
UTIME	System time (microseconds)		
ITIME	Duration (short)		
TIME	Duration		
FTIME	Duration (high resolution)		
LTIME	Duration (long)		
NTIME	Duration (nanoseconds)		
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)		
STRINGI	International character string		
SWORD	bit string - 8 bits		
WORD	bit string - 16-bits		
DWORD	bit string - 32-bits		
LWORD	bit string - 64-bits		
EPATH	Path segments		

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.

a) 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 ongoing 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(
       IDENTIFIER      id,
       DLL_config_data  params );

DLL_set_pending_confirm(
       IDENTIFIER   id,
       BOOL      result );

DLL_set_current_request(
       IDENTIFIER      id,
       DLL_config_data  params );

DLL_set_current_confirm(
       IDENTIFIER   id,
       BOOL      result );

DLL_get_pending_request(
       IDENTIFIER   id );

DLL_get_pending_confirm(
       IDENTIFIER      id,
       DLL_config_data  params );

DLL_get_current_request(
       IDENTIFIER  id );

DLL_get_current_confirm(
       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
};                      // shall 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(
       IDENTIFIER    id,
       USINT      start_count );
```

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

NOTE The tMinus fixed tag service (see 6.9) is used by the active 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 11. 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 could be incorrect.

Table 11 – 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 `ph_frame_indication` 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 `ph_frame_indication` from channel B stayed false
DLL_EV_txAbort	A transmit DLPDU was terminated with an abort sequence

DLL events	Description
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_nonconcurrence	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	`ph_lock_indication` went true then false before `ph_frame_indication` went true, but `ph_lock_indication` was not true long enough to indicate a possibly damaged DLPDU
DLL_EV_collision	`ph_frame_indication` 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 mac_ID );
```

The `mac_ID` 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 give the possibility for 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 µs ticks and shall be in the range 50 (500 µs) to 10 000

(100 ms). The beginning of the NUT shall be called the tone. At the tone, a counter that decrements every 10 µs 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 can 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 µs 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 µs 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 µs ticks. An additional 1,0 µs 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 6.5 is implementation dependent, however it can be better than 10 µs.

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 supervisor
   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.

The leap parameter should not be used in any control situations, but can 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.

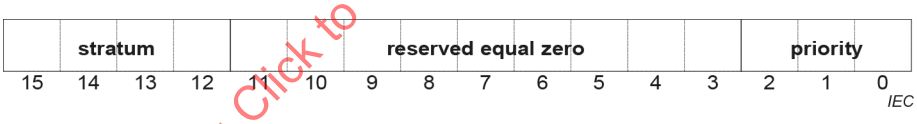


Figure 15 – Goodness parameter of TimeDist_Lpacket

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 12.

Table 12 – Time distribution priority

Value	Meaning
7	Absolute system time. Acting as supervisor
6	Absolute system time. Acting as dependent
5	Human set system time. Acting as supervisor
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 $(0,1\ \mu s)^2$. 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 µs.

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 active Keeper for the link is responsible to send 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 active 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 active 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.

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 active 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 13.

Table 13 – 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 ACTIVE_VERIFY, FAULTED_ACTIVE_VERIFY, ACTIVE, FAULTED_ACTIVE, NET_CHANGE_ACTIVE or NET_CHANGE_FAULTED_ACTIVE 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 active 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 or 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_Lpacket`s, 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 mangers 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_Lpacket`s 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`).

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.

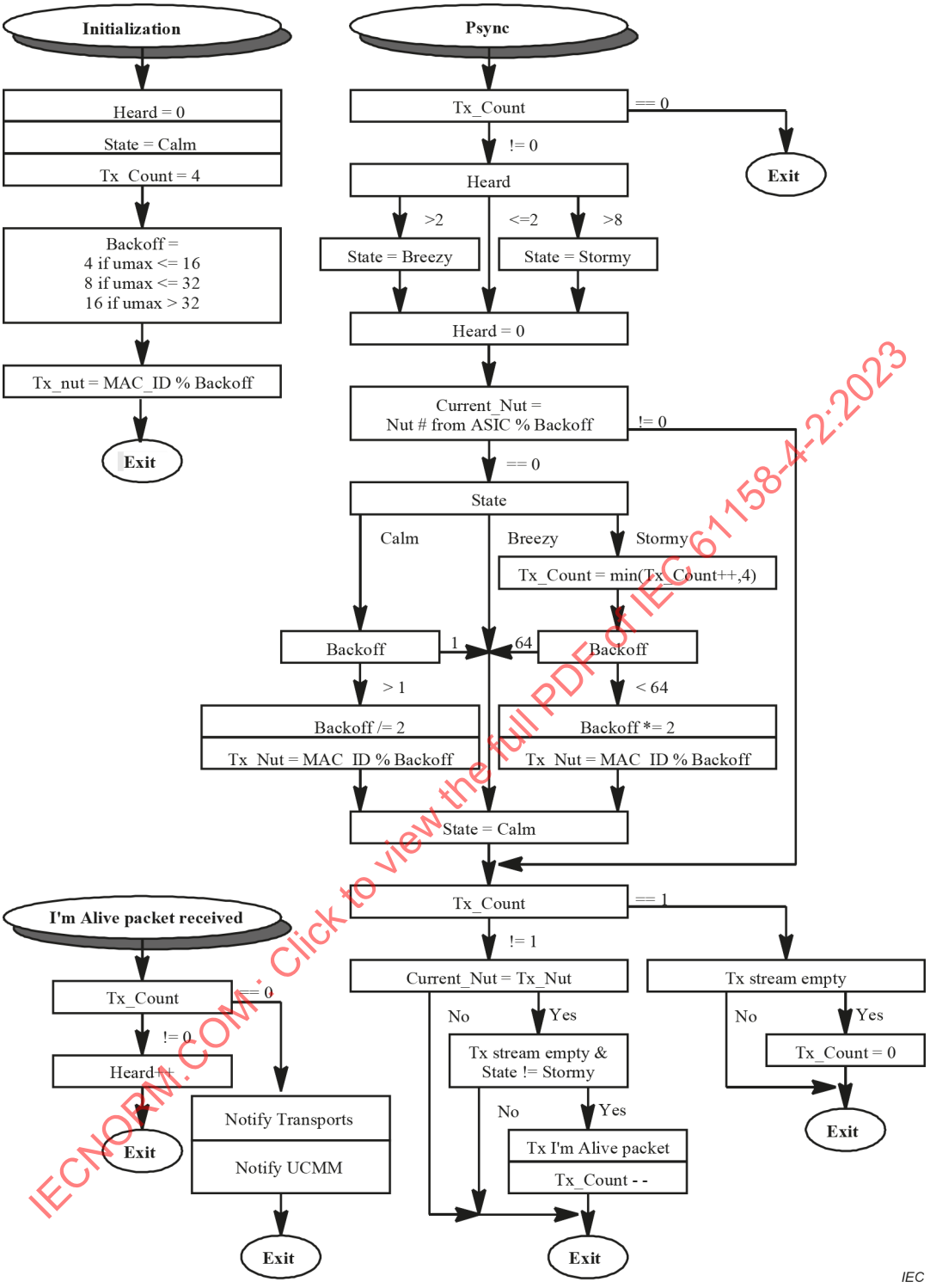


Figure 16 – Example I'm alive processing algorithm

6.11 Ping Lpackets

6.11.1 General

The ping service allows station management to initiate a ping request and receive a ping reply 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 reply 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 reply 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 reply, 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 enables 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 can 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 <MAC ID>".

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 is 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. The Keeper object also contains support for the Network Resource, a network semaphore that is used to eliminate conflicts.

The Scheduling object is used by link scheduling software (LSS) to read and write schedule information in a connection originator, and provide a signature for authentication on a 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/IEEE 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.

The Device Level Ring (DLR) object provides an interface to configure and monitor the DLR protocol.

The QoS object provides an interface to configure certain QoS-related behaviors.

The Port object describes the communication interfaces that are present on the device and visible to Type 2 networks.

The PRP/HSR Protocol object provides an interface to configure and monitor the Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR) protocol.

The PRP/HSR Nodes Table object keeps the record of all PRP or HSR capable nodes that have been detected on the network.

The LLDP Management object contains administrative information for the LLDP protocol.

The LLDP Data Table object displays a record of all adjacent LLDP implementing devices that are currently active according to the receive state machine of the LLDP protocol.

7.2 ControlNet™ [2] 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.

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 14.

Table 14 – 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
0x03 to 0x07	These class attributes are optional and are described in IEC 61158-5-2.					

7.2.3 Instance attributes

7.2.3.1 General

The ControlNet object shall support the instance attributes as specified in Table 15.

2 ControlNet™ is a trade name of ODVA, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance with a related profile does not require use of the trade name. Use of the trade name requires permission of the trade name holder.

Table 15 – 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
			reserved	USINT [16]	reserved	
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	

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
			noise_hits	USINT	noise detected that locked the modem rx PLL	
			collisions	USINT	Rx data heard just as transmission was going to start	
			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	
			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
			MAC_ver	SWORD	MAC implementation and implementation revision	see 7.2.3.12
			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 1 to 99. See 7.2.3.14
			MAC_ID_switches	USINT	MAC ID switch settings	0 to 99, 0xFF. See 7.2.3.15
			MAC_ID_changed	BOOL	MAC ID switches changed since reset	see 7.2.3.16
			Reserved	USINT	reserved	

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
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 7.2.3.17
			Reserved	USINT	reserved	
0x86	Required	Get	error_log	STRUCT of 10 octets	driver firmware buffer error counts and troublesome node list	see 7.2.3.18
			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	
			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 7.2.3.19
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 7.2.3.20
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 16, the Keeper State bit (bit 4) shall indicate whether this node has ever heard a TUI from an active Keeper. The ControlNet object shall clear this bit at initialization. Since all TUI Lpackets sent by the active 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 in 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 16 – 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 ACTIVE state 1 = TUI transmitted by Keeper in ACTIVE state
5	Keeper configured bit 0 = No active Keeper heard since joining link 1 = Heard active 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 can be read without effecting their values by using one of the Get_Attribute service requests. The information can 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 MAC_ver

The bits of the MAC_ver octet are described in Table 17.

Table 17 – Mac_ver bits

Bits	Description
0-3	**MAC revision**[a] – revision of the MAC implementation 0x1 through 0xF, 0x0 is reserved
4	Reserved – set to 0
5-7	**MAC implementation**[a] – vendor and implementation name 0 − 2 = assigned 3 = reserved – unassigned for future implementations 4 = assigned 5 − 6 = reserved – unassigned for future implementations 7 = reserved for extended MAC_ver field expansion

[a] MAC revision and MAC implementation are assigned and described by the ODVA, Inc. organization.

7.2.3.13 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 18.

NOTE The network status indicators are described in Annex A.

Table 18 – 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

7.2.3.14 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.

7.2.3.15 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.

7.2.3.16 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.

7.2.3.17 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.

7.2.3.18 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 can 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.

7.2.3.19 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.

7.2.3.20 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 19.

Table 19 – 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_Attributes_All	Get network and node specific configuration information
0x02	N/A	Optional	Set_Attributes_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_Attributes_All response

The object/service specific response data portion for a successful Get_Attributes_All 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_Attributes_All response, devices with limited resources need not support this service.

The data portion of a Get_Attributes_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 20.

Table 20 – 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 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.

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

While devices are in the listen-only state due to the Enter_Listen_Only service, they are not required to apply the Network Redundancy bits of the TUI (see Table 26), and redundant media devices may enable both channels.

NOTE 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 gives the possibility for 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 can 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_indicaton(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. Best capability means Channel A enabled for single channel devices, both channels enabled 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 active 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 a 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 active Keeper and which Keeper(s) perform backup Keeper responsibilities. The active 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 active Keeper should the active 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 21.

Table 21 – 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 22.

Table 22 – Keeper object class attributes

Number	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) (See Note 2)
0x02 to 0x07	These class attributes are optional and are described in IEC 61158-5-2.					

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 23. 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 23 – 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	
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 µs 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 µs ticks
			Blank_Time	USINT	Blanking Time	In octets
			Gb_Start	USINT	Guard Band Start	In 10 µs ticks
			Gb_Center	USINT	Guard Band Center	In 10 µs ticks
			Reserved	UINT	Reserved for Data Alignment	
			Int_Cnt mod	USINT	Interval Count Modulus (Macrocycle length)	127
			Gb_Prestart	USINT	Guard Band Prestart	In 10 µs ticks

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
0x100	Required	Get/Set	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	

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
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
			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 *co_data* attribute	
			data_size	UINT	Size of *co_data* 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 co_data parse rule supported	Level supported is ≤ Keeper object revision 0 = Format 1 1 − 255 reserved See 7.3.6.7
			0ffsets	UINT[100]	Array of word offsets into the *co_data* attribute, by MAC ID	In words 0xFFFF = no data for this MAC ID node 0 based index

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
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)
			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 24.

Table 24 – 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	Active verify
5	Active
6	Faulted Backup
7	Faulted Active verify
8	Faulted Active
9	Net Change - Backup
10	Net Change - Active
11	Net Change - Faulted Backup
12	Net Change - Faulted Active

7.3.4.3 Port status flag bit definitions

The values for the Port status flag shall be as defined in Table 25.

Table 25 – 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 26, 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 Active Keeper bit shall indicate whether or not this node has ever heard from an active Keeper. The ControlNet object shall clear this bit at initialization. All TUIs sent by the active Keeper shall have this bit set. If a TUI Lpacket is received with the Active 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 Active 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 26 – 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 ACTIVE state 1 = TUI transmitted by Keeper in ACTIVE 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 0x0101 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:

$$G(X) = X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + 1$$

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 active 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
  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;                          //       0 - 16383 by 1
   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

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 27:

Table 27 – 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 28.

Table 28 – 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:

a) obtain the Network Resource;

b) issue a Change_Start service request to the Keeper;

c) issue the appropriate Set_Attribute service request(s) to update the pending copy of the Keeper attributes;

d) issue a Change_Complete service request to the Keeper. Doing this shall initiate a synchronized parameter change;

e) 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 29.

Table 29 – 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 Active, Net Change Active, Faulted Active or Net Change Faulted Active 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 30.

Table 30 – 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 UDINT UDINT UDINT	vendor ID serial number class instance
0x4C	N/A	Required	Hold_ Network_Resource (HNR)	Continue to hold the previously obtained Network Resource for this link for 60 s	UINT UDINT UDINT UDINT	vendor ID serial number class instance
0x4D	N/A	Required	Release_ Network_Resource (RNR)	Release the Network Resource for this link	UINT UDINT UDINT UDINT	vendor ID serial number class instance
0x4E	N/A	Required	Change_Start (CS)	Copy current attributes in non-volatile storage to pending attributes in RAM. Enter the appropriate *net change* operating state	<none>	

Service code	Need in implementation		Service name	Description of service	Parameter(s)
	Class	Instance			
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 *change_abort* processing	16 bit number of "sets" performed since *Change_Start*
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 co_data 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 CO_data attribute
					UINT size; //words
					UINT offset; //offset
					UINT data [ ]

The error codes for these services shall be as specified in Table 31, where the service codes are the abbreviated terms for the service names specified in Table 30.

Table 31 – 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							

Error code	Meaning	Error code use by service								
		ONR	HNR	RNR	CS	CC	CA	GS	GAF	SAF
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 active Keeper.

A device that wishes to update the Keeper attribute information first shall submit a request to the active Keeper to obtain the NR on its behalf (via an Obtain_Network_Resource service). This request shall be broadcast since the identity of the active Keeper is not generally known. Once obtained, the device shall periodically ask (every 15 s) the active Keeper to continue holding the NR via a Hold_Network_Resource service request. The active 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 active 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 active Keeper, it shall use the data from the most recently received TUI Lpacket to hand off the holding of the NR from the old active Keeper.

In both active 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 active Keeper if the active Keeper changes. This handoff shall be invisible to the node that the active Keeper is holding the NR for. Since all information regarding the NR is contained in the TUI Lpacket being broadcast by the active 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 active Keeper, the backup Keepers shall have all the information needed to cleanly transition holding the NR when the appropriate backup Keeper takes over as active Keeper.

Faulted backup and active Keeper shall process Network Resource requests in the same way as non-faulted backup and active 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 active 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 31, 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 active 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 active 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 31, 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 active Keeper to stop holding the Network Resource if the node requesting the release is the same node for which the active 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 31, 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 active 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 31, 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 ≤ 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 response 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 31, 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 response 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 31, as designated in column CA for this service.

7.3.6.9 Get_Signature service

The Get_Signature service shall cause the active 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 31, 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.

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 31, 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 31, 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 active_verify, faulted_active_verify, active, faulted_active, net_change_active or net_change_faulted_active 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 32.

Table 32 – 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)

Name	Data type	Description of parameter	Semantics of values
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 33 lists possible error codes and the most likely condition under which the code would be returned.

Table 33 – 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 a 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	nonexistant_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 34.

Table 34 – 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 active Keeper for this link
4 – Active 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 active Keeper
5 – Active	The Keeper object has determined that it is a legitimate Keeper for this link and that it is the active 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 active 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 Active 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 active Keeper
8 – Faulted Active	The Keeper object is faulted and has assumed faulted active 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 active Keeper
9,10,11,12 – Net Change	The Keeper is having its attributes updated. There is a Net *Change* substate for the *backup, active, fault* and *faulted active* states previously discussed. The separate substates are needed so the Keeper object can exit the *Net Change* state properly when updates complete normally and when updates are aborted. The pending attributes may be set only when the Keeper is in a *Net Change* state. Nodes in the *net change active* and *net change faulted active* shall respond to requests of the active Keeper. Nodes in the *net change backup* or *net change faulted backup* states shall not respond to requests of the active 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 active Keeper shall be the Keeper with the lowest MAC ID of all the legitimate Keepers on the link. A faulted active Keeper on a link can have the lowest MAC ID or another MAC ID.

The Keeper shall be responsible for configuring those devices that appear in its attributes. The active 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 active 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 active 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 active 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.

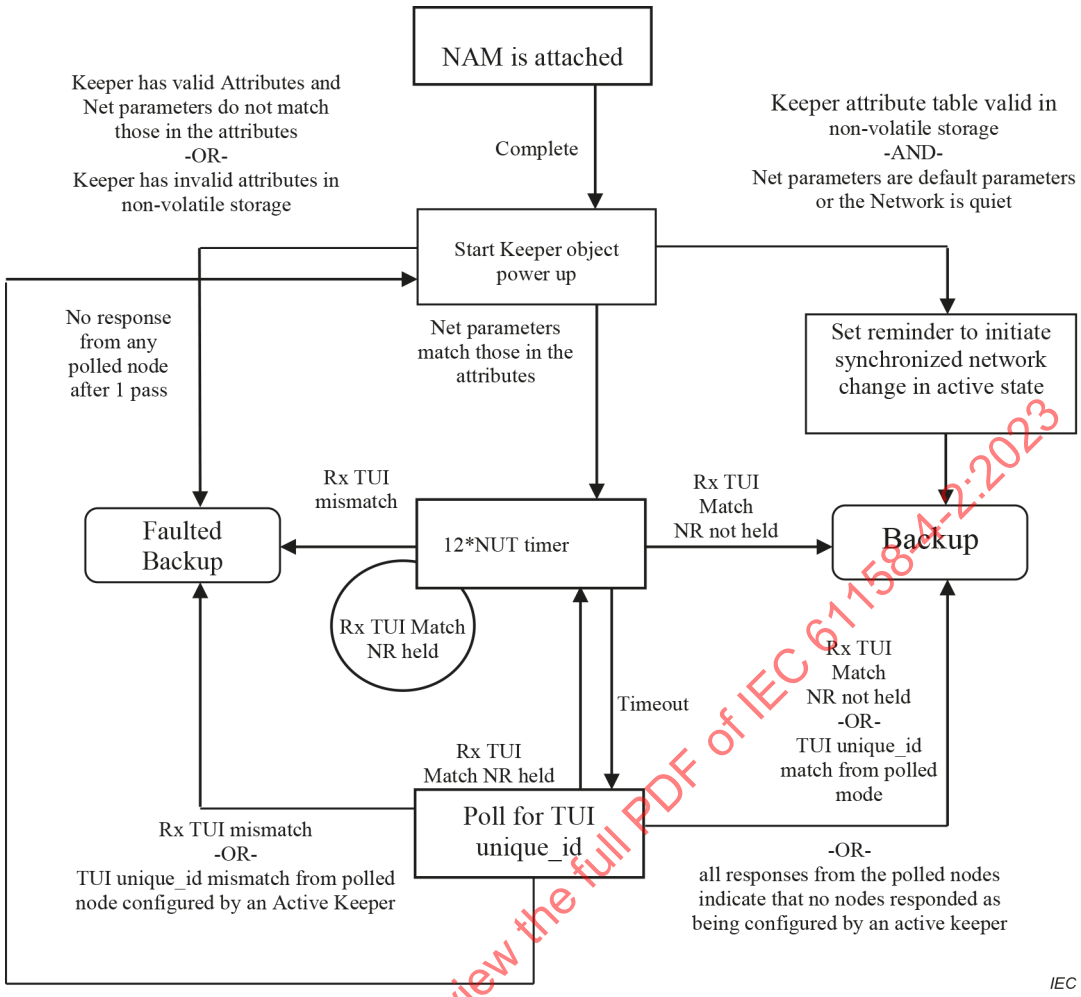


Figure 18 – Keeper object power-up state diagram

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:

a) 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.

b) 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.

c) 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.

d) 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 "Active" 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 "Active" 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 "Active" 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 35, and in 7.3.10.3 to 7.3.10.9.

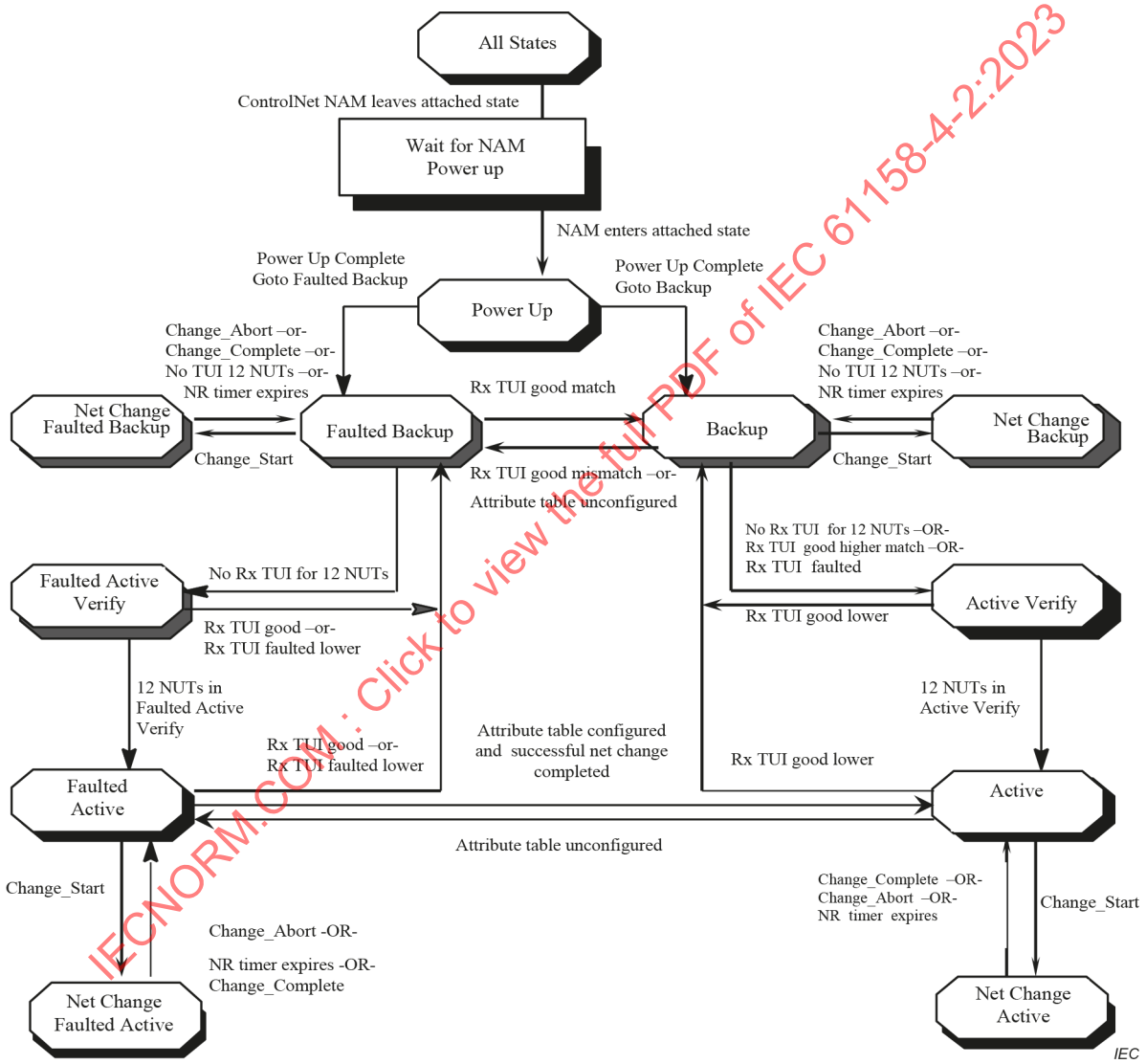


Figure 19 – Keeper object operating state diagram

Table 35 – Keeper object state event matrix

Event	State [a]										
	Power Up	Backup	Active Verify	Active	Faulted Backup	Faulted Active Verify	Faulted Active	Net Change (Back-up)	Net Change (Active)	Net Change (Faulted Backup)	Net Change (Faulted Active)
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 Active 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 Active Verify									
Rx TUI from a faulted lower node						Goto Faulted Backup					
No TUI Rx for 12*NUT		Goto Active Verify			Goto Faulted Active Verify			Abort change. Goto Backup		Abort change. Goto Faulted Backup	
12 Nut times elapse			Goto Active			Goto Faulted Active					
Change_Start		Goto Net Change Backup		Goto Net Change Active	Goto Net Change Faulted Backup		Goto Net Change Faulted Active				
Change_Complete								Goto Backup	Goto Active	Goto Backup	Goto Active verify
Change_Abort								Do Change Abort. Goto Backup	Do Change Abort. Goto Active	Do Change Abort. Goto Faulted Backup	Do Change Abort Goto Faulted Active
60 s Network Resource timer expires								Abort change. Goto Backup	Abort change. Goto Active	Abort change. Goto Faulted Backup	Abort change. Goto Faulted Active
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										

[a] 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 active Keeper for the node performing the attribute updates. If the active Keeper does not hear from the node performing the update at least once every 60 s, the active 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 active 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 active Keeper.

7.3.10.7 Active verify

The Keeper shall wait for 12*NUT after the first TUI broadcast begins before assuming active Keeper status. This shall be sufficient time for a Keeper with a lower MAC ID to become active Keeper. Other Keepers become backup Keepers.

7.3.10.8 Rogue

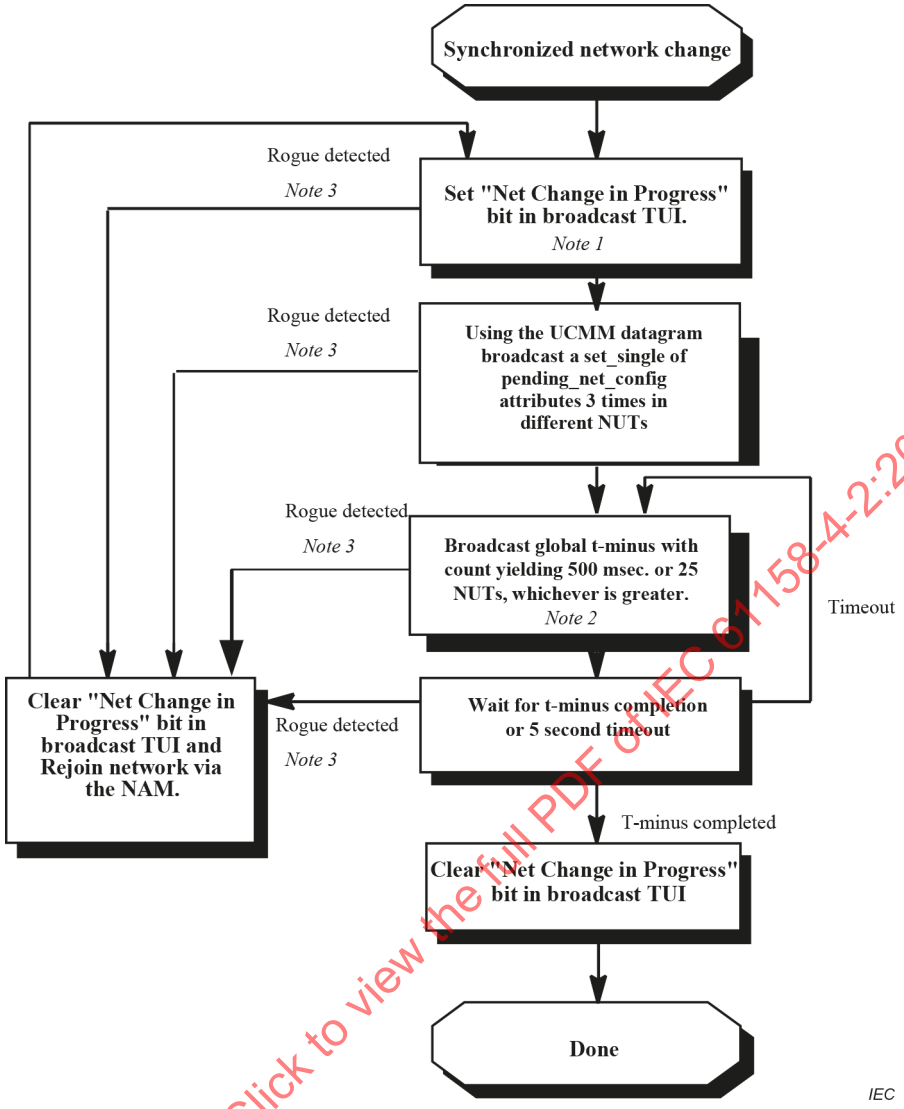
If the node containing the Keeper rogues during a synchronized network change sequence, the "Net Change in Progress" bit shall be cleared and control shall be returned back to the start of the synchronized network change processing immediately after the NAM brings the node back on-line. The normal Keeper 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 provides to nodes a reasonable amount of time to process the Set_Attribute_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

a) read scheduled connection information;

b) write schedule information;

c) 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 36. The Get_Attributes_All service shall return the class level attributes in numerical order, smallest to largest.

Table 36 – 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
0x04 to 0x07	These class attributes are optional and are described in IEC 61158-5-2.				

7.4.3 Instance attributes

The Scheduling object shall support the instance attributes as specified in Table 37. The Get_Attributes_All service shall return the attribute level attributes in numerical order, smallest to largest.

Table 37 – 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 µs (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 7.4.4, the Scheduling object shall support the standard Get_Attributes_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 38.

Table 38 – Scheduling object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Required	Optional	Get_Attributes_All	Returns a predefined listing of this object's attributes
0x02	N/A	Optional	Set_Attributes_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 µs.

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 39.

Table 39 – 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
0xDO	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 40.

Table 40 – 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 41.

Table 41 – 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 40.

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 response 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 response 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 response.

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 42.

Table 42 – 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 be in any 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 43.

Table 43 – 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 44.

Table 44 – 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 45.

Table 45 – 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 46.

Table 46 – 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 gives the possibility for 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 47.

Table 47 – 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 48.

Table 48 – 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
0xDO	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 can 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 network 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 can be associated with any of the following: an Ethernet ISO/IEC/IEEE 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 Type 2 port on the device.

7.5.2 Revision history

Table 49 shows the revision history for the TCP/IP Interface object.

Table 49 – Revision history

Revision	Reason for object definition update
1	Initial revision of this object definition
2	Added ACD instance attributes 10 (SelectACD) and 11 (LastConflictDetected) Added instance attribute 12, QuickConnect Added bits 5 (Interface Configuration Pending) and 6 (AcdStatus) to attribute 1, Status Added bits 6 (Interface Configuration Change Requires Reset) and 7 (AcdCapable) to attribute 2, Configuration Capability
3	Added bit 7 (AcdFault) to attribute 1, Status
4	Added instance attribute 13, Encapsulation Inactivity Timeout

7.5.3 Class attributes

The TCP/IP Interface object shall support the class attributes as specified in Table 50.

Table 50 – TCP/IP Interface object class attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	NV	Revision	UINT	Revision of this object	Third revision, value = 4
2	Conditional[a]	Get	NV	Max instance	UINT	Maximum instance number of an object currently created in this class level of the device	The largest instance number of a created object at this class hierarchy level.
3	Conditional[a]	Get	NV	Number of instances	UINT	Number of object instances currently created at this class level of the device	The number of object instances at this class hierarchy level
4 to 7	These class attributes are optional and are described in IEC 61158-5-2.						
[a] Required if the number of instances is greater than 1.							

7.5.4 Instance attributes

7.5.4.1 General

The TCP/IP Interface object shall support the instance attributes as specified in Table 51.

Table 51 – TCP/IP Interface object instance attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	V	Status	DWORD	Interface status	See 7.5.4.2
2	Required	Get	NV	Configuration capability	DWORD	Interface capability flags	Bit map of capability flags. See 7.5.4.3
3	Required	Get Set is conditional[a]	NV	Configuration control	DWORD	Interface control flags	Bit map of control flags. See 7.5.4.4
4	Required	Get	NV	Physical link object	STRUCT of variable size	Path to physical link object	See 7.5.4.5
				Path size	UINT	Size of path	Number of UINTs in Path
				Path	Padded EPATH	Logical segments identifying the physical link object	The path is restricted to one logical class segment and one logical instance segment. The maximum size is 12 octets.
5	Required	Get Set is recommended	NV when configuration method is 0. V when obtained via BOOTP or DHCP	Interface configuration	STRUCT of variable size	TCP/IP network interface configuration	See 7.5.4.6

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
				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

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
				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
				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
6	Required	Get Set is conditional[b]	NV	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 7.5.4.6.2
7	Conditional[c]			Safety network number	USINT[6]		See IEC 61784-3-2
8	Conditional[d]	Get Set is conditional[e]	NV	TTL value	USINT	TTL value for Type 2 Ethernet multicast packets	Time-to-live value for IP multicast packets. Default value is 1, minimum is 1, maximum is 255. See 7.5.4.8

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
9	Conditional[d]	Get Set is conditional[e]	NV	Mcast config	STRUCT of 8 octets	IP multicast address configuration	See 7.5.4.9
				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 7.5.4.9 for details
				Reserved	USINT	Reserved for future use	Shall be 0
				Num mcast	UINT	Number of IP multicast addresses to allocate for Type 2 Ethernet	The number of IP multicast addresses allocated, starting at "Mcast start addr". Maximum value is device specific, however shall not exceed the number of Type 2 Ethernet 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
10	Conditional[f]	Set	NV	SelectAcd	BOOL	Activates the use of ACD	Enable ACD (1, default), disable ACD (0). See 7.5.4.10
11	Conditional[f]	Set	NV	LastConflictDetected	STRUCT of 35 octets	Structure containing information related to the last conflict detected	ACD Diagnostic Parameters See 7.5.4.11
				AcdActivity	USINT	State of ACD activity when last conflict detected	ACD activity Default = 0
				RemoteMAC	USINT[6]	MAC address of remote node from the ARP PDU in which a conflict was detected	MAC from Ethernet packet header Default =0
				ArpPdu	USINT[28]	Copy of the raw ARP PDU in which a conflict was detected.	ARP PDU Default = 0

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
12	Optional	Set	NV	QuickConnect	BOOL	Enable/Disable of QuickConnect feature	0 = Disable (default) 1 = Enable See 7.5.4.12
13	Conditional[g]	Set	NV	Encapsulation Inactivity Timeout	UINT	Number of seconds of inactivity before TCP connection is closed	0 = Disable 1 to3 600 = timeout in seconds Default = 120 See 7.5.4.13
14	Conditional[h]	Get	NV	IANA Port Admin	STRUCT of variable size	IANA port admin configuration	See 7.5.4.14
					USINT	Port count	Number of elements
					ARRAY of STRUCT	Port Array	
					SHORT_STRING	Port Name	Name of the port
					UINT	Port Number	IANA port number
					USINT	Protocol	6 = TCP 17 = UDP
					BOOL	Admin State	0 = Closed 1 = Open
					SWORD	Admin Capability	Capability about Port Array entry
15	Optional	Get	NV	IANA Protocol Admin	STRUCT of variable size	IANA protocol admin configuration	See 7.5.4.15
					USINT	Protocol count	Number of elements
					ARRAY of STRUCT	Protocol Array	
					SHORT_STRING	Protocol Name	Name of the protocol
					USINT	Protocol Number	IANA protocol number
					BOOL	Admin State	0 = Disabled 1 = Enabled
					SWORD	Admin Capability	Capability about Protocol Array entry
16	Optional	Get	V	Active TCP Connections	UINT	Current count of active TCP connections in use by Type 2	See 7.5.4.16

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
17	Optional	Get	V	Non-Type 2 Encapsulation Messages Per Second	UDINT	Number of non-Type 2 encapsulation messages sent and received by this device over the last second	See 7.5.4.17

NOTE ACD is specified in 7.5.9.

[a] Set is required unless the configuration method is selected exclusively via hardware settings.

[b] Set is optional when the Interface Configuration attribute is not settable.

[c] This attribute is required for Type 2 safety devices. Non-safety devices shall not implement this attribute.

[d] If either "TTL value" or "Mcast config" is implemented, both shall be implemented.

[e] If either "TTL value" or "Mcast config" is implemented as settable, both shall be implemented as settable.

[f] Required if device implements ACD.

[g] Required only if supported by the Type 2 Ethernet transport profile, otherwise it is not allowed.

[h] Optional by default (only required if the device supports Type 2 over (D)TLS, which is outside the scope of this document).

7.5.4.2 Status

The status attribute is a bitmap that indicates the status of the TCP/IP network interface, as specified in Table 52. Refer to the state diagram in Figure 21 for a description of object states as they relate to the status attribute.

Table 52 – 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 IP address member of 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	Interface configuration pending	Indicates a pending configuration change in the Interface Configuration attribute. This bit shall be 1 (TRUE) when Interface Configuration attribute are set and the device requires a reset in order for the configuration change to take effect (as indicated in the Configuration Capability attribute). The intent of the Interface Configuration Pending bit is to allow client software to detect that a device's IP configuration has changed, but will not take effect until the device is reset.
6	AcdStatus	Indicates when an IP address conflict has been detected by ACD. This bit shall default to 0 (FALSE) on startup. If ACD is supported and enabled, then this bit shall be set to 1 (TRUE) any time an address conflict is detected as defined by the [ConflictDetected] transitions in Figure 23 (ACD Behavior).

Bit(s)	Name	Definition
7	AcdFault	Indicates when an IP address conflict has been detected by ACD or the defense failed, and that the current Interface Configuration cannot be used due to this conflict. This bit SHALL be 1 (TRUE) if an address conflict has been detected and this interface is currently in the Notification & FaultAction or AcquireNewIpv4Parameters ACD state as defined in 7.5.9, and SHALL be 0 (FALSE) otherwise. Notice that when this bit is set, then this Type 2 port will not be usable. However, for devices with multiple ports, this bit provides a way of determining if the port has an ACD fault and thus cannot be used.
8	IANA Port Admin Change Pending	Indicates a pending configuration change in the IANA Port Admin attribute. This bit shall be set when the device requires a reset in order for the configuration change to take effect (as indicated in the Admin Capability element of the IANA Port Admin attribute).
9	IANA Protocol Admin Change Pending	Indicates a pending configuration change in the IANA Protocol Admin attribute. This bit shall be set when the device requires a reset in order for the configuration change to take effect (as indicated in the Admin Capability element of the IANA Protocol Admin attribute).
10-31	Reserved	Reserved for future use and shall be set to zero

7.5.4.3 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 53. 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 53 – 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	Shall be 0, behavior to be defined in a future edition of this document
4	Configuration settable	1 (TRUE) shall indicate the interface configuration attribute is settable. Some devices, for example a PC or workstation, may prevent the interface configuration to be set via the TCP/IP Interface object
5	Hardware configurable	1 (TRUE) shall indicate the IP Address member of the Interface Configuration attribute can be obtained from hardware settings (e.g., pushwheel, thumbwheel,…). If this bit is FALSE the Status instance attribute (1), Interface Configuration Status field value shall never be 2 (the Interface Configuration attribute contains valid configuration, obtained from hardware settings)
6	Interface configuration change requires reset	1 (TRUE) shall indicate that the device requires a restart in order for a change to the Interface Configuration attribute to take effect. If this bit is FALSE a change in the Interface Configuration attribute will take effect immediately.
7	AcdCapable	(1) TRUE shall indicate that the device is ACD capable
8-31	Reserved	Reserved for future use and shall be set to zero

7.5.4.4 Configuration control

7.5.4.4.1 Configuration control structure

The configuration control attribute is a bitmap used to control network configuration options, as specified in Table 54.

Table 54 – Configuration control bits

Bit	Name	Definition	
0–3	Configuration method	Determines how the device shall obtain its IP-related configuration	0 = The device shall use statically-assigned IP configuration values 1 = The device shall obtain its interface configuration values via BOOTP. 2 = The device shall obtain its interface configuration values via DHCP 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	

7.5.4.4.2 Configuration method

The Configuration Method determines how a device shall obtain its IP-related configuration.

- If the Configuration Method is 0, the device shall use statically-assigned IP configuration contained in the Interface Configuration attribute (or assigned via non-Type 2 methods, as noted below).
- If the Configuration Method is 1, the device shall obtain its IP configuration via BOOTP. The BOOTP client behavior shall be as defined in the relevant IETF RFCs (IETF RFC 951, IETF RFC 1542, IETF RFC 2132, or their successors).
- If the Configuration Method is 2, the device shall obtain its IP configuration via DHCP. The DHCP client behavior shall be as defined in the relevant IETF RFCs (IETF RFC 2131, IETF RFC 2132, or their successors).
- Devices that optionally provide hardware means (e.g., rotary switch) to configure IP addressing behavior shall set the Configuration Method to reflect the configuration set via hardware: 0 if a static IP address has been configured, 1 if BOOTP has been configured, 2 if DHCP has been configured.

If a device has been configured to obtain its configuration via BOOTP or DHCP, it shall continue sending requests until a response from the server is received. Devices that elect to use default IP configuration in the event of no response from the server shall continue issuing requests until a response is received, or until the Configuration Method is changed to static.

Once the device receives a response from the server, it shall stop sending the BOOTP/DHCP client requests (DHCP clients shall follow the lease renewal behavior per the IETF RFC). It is recommended that devices implement the means to detect a link up and upon a link up detection, restart the initial BOOTP or DHCP sequence. For multiport devices, the restart of the initial BOOTP or DHCP sequence shall only be triggered if all external links have been down and when the first link up is detected.

Setting the Configuration Method to 0 (static address) shall cause the Interface Configuration to be saved to non volatile storage.

It is recommended that setting the Configuration Method to 1 (BOOTP) or 2 (DHCP) cause the device to start the BOOTP / DHCP client to obtain new IP address configuration. If the device requires a reset in order to start the BOOTP / DHCP client, it shall set the Interface Configuration Pending bit, and upon device reset, start the BOOTP / DHCP client.

7.5.4.4.3 DNS enable

For originator devices that support resolving target host names via DNS, the DNS Enable bit shall enable (1) and disable (0) the DNS client.

7.5.4.5 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 two logical segments (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).

The physical link object itself typically maintains link-specific counters as well as any link-specific configuration attributes. If the port associated with the TCP/IP Interface object has an Ethernet physical layer, this attribute shall point to an instance of the Ethernet Link object (see 7.6). When there are multiple physical interfaces that correspond to the TCP/IP interface, this attribute shall either contain a path size of 0, or shall contain a path to the object representing an internal communications interface (often used in the case of an embedded switch).

An example path is shown in Table 55 (values are hexadecimal).

Table 55 – 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

7.5.4.6 Interface configuration

7.5.4.6.1 Interface configuration contents

The Interface Configuration attribute contains the configuration parameters required for a device to operate as a TCP/IP node.

The contents of the Interface Configuration attribute shall depend upon how the device has been configured to obtain its IP parameters.

- If configured to use a static IP address (Configuration Method value is 0), the Interface Configuration values shall be those which have been statically assigned and stored in non-volatile storage.
- If configured to use BOOTP or DHCP (Configuration Method value is 1 or 2), the Interface Configuration values shall contain the configuration obtained from the BOOTP or DHCP server. The Interface Configuration attribute shall be 0 until the BOOTP/DHCP reply is received.
- Some devices optionally provide additional, non-Type 2 mechanisms for setting IP-related configuration (e.g., a web server interface, rotary switch for configuring IP address,…). When such a mechanism is used, the Interface Configuration attribute shall reflect the IP configuration values in use.

Components of the interface configuration attributes are described in Table 56.

Table 56 – 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 fieldbus connection 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"

7.5.4.6.2 Set Attributes behavior

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, client software should first Get the Interface Configuration attribute, change the desired parameters, and then Set the attribute.

An attempt to set any of the parameters of the Interface Configuration attribute to invalid values (see Table 51) shall result in an error response with status code 0x09 "Invalid Attribute Value" to be returned. In this scenario, all of the parameters of the Interface Configuration attribute retain the values that existed prior to the invocation of the set service.

If the device has an active I/O connection, it is recommended that the device rejects the set attributes request by returning an error response with status code 0x10 "Device State Conflict".

When the value of the Configuration Method (Configuration Control attribute) is 0, the Set Attribute service shall store the new Interface Configuration values in non-volatile memory. If the device requires reset in order for new parameters to take effect (Configuration Capability bit 6 set), the device shall set the Interface Configuration Pending bit (Status attribute bit 5).

After storing the new values the device shall send a response using its current IP address (i.e., the IP address to which the set attributes request was sent).

If the device does not require reset (Configuration Capability bit 6 clear) in order for new IP configuration to take effect, after responding to the set service, the device shall initiate application of the new IP parameters. While implementation-dependent, this activity is generally initiated by the TCP/IP Interface Object set service and then completed asynchronously by the TCP/IP stack.

In order to achieve consistency of device configuration behavior, it is recommended that devices support setting the Interface Configuration attribute, and support application of new attribute values without requiring device reset. If a device does not support setting the Interface Configuration attribute, the device shall return an error response with status code 0x0E "Attribute Not Settable".

7.5.4.6.3 Application of new configuration parameters

If the device does not require reset (Configuration Capability bit 6 clear), after responding to the set service the device shall initiate application of the new IP parameters. When applying new IP configuration, the device shall maintain the consistency of the IP configuration context in which it operates. For example, the device shall not mix old and new IP address/network mask/gateway address values, and shall not use the new IP configuration on Type 2 connections established with the previous IP address.

When a TCP/IP stack is configured to use a particular set of IP parameters, a context around these IP parameters is built. This context includes relationships to the TCP/IP stack, the Type 2 communications environment, and the control application among other entities. To allow a different set of IP parameters to be used, a new IP context shall be built. The device shall properly manage its IP context and maintain its consistency. For example a new IP Address shall not be used with old Network Mask or Gateway Address; an old IP address shall not be used for Type 2 or other communication once the new one is applied.

7.5.4.7 Host name

The Host Name attribute contains the device's host name, which can be used for informational purposes. The set access is optional when the Interface Configuration attribute is not settable.

7.5.4.8 TTL value

TTL value is the value the device shall use for the IP header Time-to-live field when sending Type 2 Ethernet 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. If a device does not support applying the TTL Value immediately, the Mcast pending bit in the Interface status attribute shall be set, indicating that there is pending configuration. For devices that support applying the TTL Value immediately, if there are existing multipoint connections, an Object State Conflict error (0xC) shall be returned and the Mcast Pending bit shall not be set. When a new TTL value is pending, Get_Attribute_Single or Get_Attributes_All requests shall return the pending value. The Mcast pending bit shall be cleared the next time the device starts.

Users should exercise caution when setting the TTL value greater than 1, to prevent unwanted multicast traffic from propagating through the network.

NOTE IEC 61158-6-2 includes a discussion on user considerations when using multicast.

7.5.4.9 Mcast config

The Mcast config attribute contains the configuration of the device's IP multicast addresses to be used for Type 2 Ethernet 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 57 shows the details for alloc control.

Table 57 – 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_Attributes_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 Type 2 Ethernet 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. If a device does not support applying the MCast Config attribute immediately, the Mcast Pending bit in the Interface status attribute shall be set, indicating that there is pending configuration. For devices that support applying the Mcast Config attribute immediately, if there are existing multipoint connections, an Object State Conflict error (0xC) shall be returned and the MCast Pending bit shall not be set. When a new Mcast Config value is pending, Get_Attribute_Single or Get_Attributes_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.

7.5.4.10 SelectAcd

SelectAcd is an attribute used to Enable/Disable ACD.

If SelectAcd is 0 then ACD is disabled. If SelectAcd =1 then ACD is enabled.

The default value of SelectAcd shall be 1 indicating that ACD is enabled.

When the value of SelectAcd is changed by a Set_Attribute service, the new value of SelectAcd shall not be applied until the device executes a restart.

7.5.4.11 LastConflictDetected

The LastConflictDetected attribute is a diagnostic attribute presenting information about the ACD state when the last IP Address conflict was detected. This attribute shall be updated by the device whenever an incoming ARP packet is received that represents a conflict with the device's IP address as described in IETF RFC 5227.

To reset this attribute the Set_Attribute_Single service is invoked with an attribute value of all 0. Values other than 0 shall result in an error response (status code 0x09, Invalid Attribute Value).

There are three elements to the LastConflictDetected structure: AcdActivity, RemoteMac and ArpPdu.

- AcdActivity contains the state of the ACD algorithm when the last IP address conflict was detected. The ACD activities are defined in Table 58.

Table 58 – AcdActivity values

Value	AcdMode	Description
0	NoConflictDetected (Default)	No conflict has been detected since this attribute was last cleared
1	ProbeIpv4Address	Last conflict detected during ProbeIpv4Address state
2	OngoingDetection	Last conflict detected during OngoingDetection state or subsequent DefendWithPolicyB state
3	SemiActiveProbe	Last conflict detected during SemiActiveProbe state or subsequent DefendWithPolicyB state

- RemoteMac contains the ISO/IEC/IEEE 8802-3 source MAC address from the header of the received Ethernet packet which was sent by a device reporting a conflict.
- ArpPdu contains the ARP Response PDU in binary format. The ArpPdu shall be a copy of the ARP message that caused the address conflict. It shall be a raw copy of the ARP message as it appears on the Ethernet network, i.e.: ArpPdu[1] contains the first octet of the ArpPdu received (see Table 59).

Table 59 – ArpPdu – ARP Response PDU in binary format

Field size (octets)	Field description	Field value
2	Hardware Address Type	1 for Ethernet H/W
2	Protocol Address Type	0x800 for IP
1	HADDR LEN	6 for Ethernet H/W
1	PADDR LEN	4 for IP
2	OPERATION	1 for Request or 2 for Response
6	SENDER HADDR	Sender's H/W address
4	SENDER PADDR	Sender's proto address
6	TARGET HADDR	Target's H/W address
4	TARGET PADDR	Target's proto address

7.5.4.12 QuickConnect™ [3]

The QuickConnect attribute shall enable (1) or disable (0) the QuickConnect feature. The default value of the attribute shall be 0.

NOTE The QuickConnect™ power up implementation profile is specified by the ODVA, Inc organization.

7.5.4.13 Encapsulation Inactivity Timeout

The Encapsulation Inactivity Timeout attribute is used to enable TCP socket cleanup (closing) when the defined number of seconds have elapsed with no Encapsulation activity. Encapsulation activity is defined in IEC 61158-6-2, 11.7.3.2.

When set, the Encapsulation Inactivity Timeout attribute shall be saved in non-volatile memory and applied to all subsequently-opened connections. Devices should also, if feasible, apply Encapsulation Inactivity Timeout changes to all currently-opened connections.

---

[3] QuickConnect™ is a trade name of ODVA, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance with this profile does not require use of the trade name QuickConnect™. Use of the trade name QuickConnect™ requires permission from ODVA, Inc.

7.5.4.14 IANA Port Admin

The TCP and UDP port numbers are a part of the transport layer. The port numbers are used to identify the sending and receiving applications within the communicating devices. Port numbers are divided into three ranges: well-known, registered, and dynamic or private. The registered ports are assigned by Internet Assigned Numbers Authority (IANA), www.iana.org. Both well-known and registered ports are listed in the Service Name and Transport Protocol Port Number Registry at IANA.

The IANA Port Admin attribute lists TCP and UDP ports used by the device where it acts as a server. It is recommended that all TCP and UDP ports used by the device be exposed by the attribute. At a minimum all Type 2-related ports supported by the devices shall be exposed.

The text provided in the Port Name member is vendor specific but for well-known and registered ports it is recommended to use the IANA description. It is valid to have a NULL string in the Port Name attribute.

The state (open or closed) is indicated in the Admin State member. If the state has been changed using the Set_Port_Admin_State service (see 7.5.7.2) and the device requires a reset in order for the changes to take effect, the pending values shall be returned for the Admin State member until the module has been restarted. The default state for each port is vendor specific but it is recommended that all Type 2 related ports are open.

The Admin Capability member is a bit array that holds information about the Port Array entry, as specified in Table 60.

Table 60 – Admin Capability member bit definitions

Bit(s):	Called	Definition
0	Configurable	Indicates if the Admin State member can be changed using the Set_Port_Admin_State service (see 7.5.7.2).
1	Reset Required	1 (TRUE) shall indicate that the device requires a restart in order for a change to the Admin State member to take effect. If this bit is FALSE a change to the Admin State member will take effect immediately. Different Port Array members may have different values in this bit.
2-7	Reserved	Reserved for future use and shall be set to zero.

When a port state has been set to close, the TCP/IP interface shall behave as if the associated server was not running.

NOTE For a device with more than one TCP/IP Interface this allows a user to close/open an IANA port on one TCP/IP Interface and configure the opposite behavior on a separate TCP/IP Interface.

The behavior when receiving a messaged to a closed port differs between TCP/IP implementations, for example the message could silently be dropped or a message indicating that the port is closed could be returned. However the behavior for closed port shall be identical to ports not used within the device.

The sequence of operation when changing the attribute would be as follows.

1) The client reads the IANA Port Admin attribute and presents the list of ports that can be configured.
2) The user changes the Admin State of one of the ports in the list.
3) If the Reset Required bit is set in the Admin Capability member the user will be informed by the client that a reset is required for the change to take effect.
4) After storing the new setting the device shall send response back to the client.

5) If the Reset Required bit in the Admin Capability member is FALSE (0) the device will either open or close the port immediately.

6) If the Reset Required bit in the Admin Capability member is TRUE (1) the device will set the IANA Port Admin Change Pending bit in the State attribute, and will either open or close the port the next time the device restarts.

7.5.4.15 IANA Protocol Admin

This attribute lists all IANA protocols used in the device, e.g. ICMP. The protocol is the same as the protocol field within the IPv4 header, and is used to define the protocol that is carried by the data portion of the IP datagram. IANA maintains a list of all IP protocol numbers.

Any IANA listed protocol can be included and the list can therefore contain all available protocols within the device. It is recommended that a device expose all IANA protocols implemented but a particular implementation may elect not to do so.

The text provided in the Protocol Name member is vendor specific but it is recommended to use the IANA description. It is valid to have a NULL string in the Protocol Name attribute.

If the state has been changed using the Set_Protocol_Admin_State service (see 7.5.7.3) and the device requires a reset in order for the changes to take effect, the pending values shall be returned for the Admin State member until the module has been restarted. The default state (enabled or disabled) for each protocol is vendor specific.

The Admin Capability member is a bit array that holds information about the Protocol Array entry, as specified in Table 61.

Table 61 – Admin Capability member bit definitions

Bit(s):	Called:	Definition
0	Configurable	Indicates if the Admin State member can be changed using the Set_Protocol_Admin_State service (see 7.5.7.3).
1	Reset Required	1 (TRUE) shall indicate that the device requires a restart in order for a change to the Admin State member to take effect. If this bit is FALSE, a change to the Admin State member will take effect immediately. Different Protocol Array members might have different values in this bit.
2-7	Reserved	Reserved for future use and shall be set to zero.

When a protocol state has been set to disabled, the TCP/IP interface shall behave as if the associated server was not running, thus blocking any incoming requests on this protocol.

NOTE For a device with more than one TCP/IP Interface this allows a user to enable/disable an IANA protocol on one TCP/IP Interface and configure the opposite behavior on a separate TCP/IP Interface.

The sequence of operation when changing the attribute would be as follows.

1) The client reads the IANA Protocol Admin attribute and presents the list of protocols that can be configured.

2) The user changes the Admin State of one of the ports in the list.

3) If the Reset Required bit is set in the Admin Capability member the user will be informed by the client that a reset is required for the change to take effect.

4) After storing the new setting the device shall send response back to the client.

5) If the Reset Required bit in the Admin Capability member is FALSE (0) the device will either open or close the port immediately.

6) If the Reset Required bit in the Admin Capability member is TRUE (1) the device will set the IANA Protocol Admin Change Pending bit in the State attribute, and will either open or close the port the next time the device restarts.

7.5.4.16 Active TCP Connections

This value holds the current number of TCP connections that are in-use by Type 2 for the IP address associated with this TCP/IP Interface object instance. This includes inbound and outbound TCP/IP connections.

7.5.4.17 Non-Type 2 Encapsulation Messages per Second

The Non-Type 2 Encapsulation Messages per Second attribute indicates the total number of encapsulation commands, excluding SendUnitData and SendRRData, sent and received by the IP address associated with this TCP/IP Interface object instance in the previous second.

NOTE Explicit messages sent using SendUnitData and SendRRData messages would be counted via the Connection Manager attribute for explicit message rate (see Connection Manager definition in IEC 61158-5-2 and IEC 61158-6-2).

7.5.5 Diagnostic connection points

One diagnostic connection point is defined for the TCP/IP Interface object class at the instance level. This connection point is required if the Standard Network Diagnostic Assembly is implemented (see IEC 61158-6-2, 4.1.8.4.1.4).

NOTE See IEC 61158-5-2, 6.1.6 for further information about Diagnostic Connection Points.

The format of the TCP/IP Interface diagnostic connection point is as specified in Table 62.

Table 62 – TCP/IP Interface connection point 1, Standard Network Diagnostics

Attribute ID	Attribute name	Data type	Attribute size	Size of structure
17	Non-Type 2 Encapsulation Messages Per Second	UDINT	4 octets	8 octets
16	Active TCP Connections	UINT	2 octets	
N/A	16 bits pad, shall be zeros		2 octets	

7.5.6 Common services

7.5.6.1 General

The TCP/IP Interface object shall support the common services as specified in Table 63.

Table 63 – TCP/IP Interface object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (see the Get_Attributes_All response definition in 7.5.6.2)
0x02	N/A	Optional	Set_Attributes_All	Modifies all settable attributes
0x0E	Required	Required	Get_Attribute_Single[a]	Returns the contents of the specified attribute or connection point
0x10	N/A	Required	Set_Attribute_Single	Modifies a single attribute
0x1D	N/A	Conditional[b]	Get_Connection_Point_Member_List	Used to get member paths of a Connection Point structure

[a] This service shall support the use of a connection point in the service path when Diagnostic Connection Point(s) are implemented (see 7.5.5).

[b] This service is required if Diagnostic Connection Point(s) are implemented.

7.5.6.2 Get_Attributes_All response

At the class level, the Get_Attributes_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

For instance attributes, attributes shall be returned in numerical order up to the last implemented attribute. The Get_Attributes_All response shall be as specified in Table 64.

Table 64 – Get_Attributes_All response format

Attribute ID	Data Type	Attribute Name	Default Value (if not implemented)
1	DWORD	Status	
2	DWORD	Configuration Capability	
3	DWORD	Configuration Control	
4	STRUCT of	Physical Link Object	
	UINT	Path Size	
	Padded EPATH	Path	
5	STRUCT of	Interface Configuration	
	UDINT	IP Address	
	UDINT	Network Mask	
	UDINT	Gateway Address	
	UDINT	Name Server	
	UDINT	Name Server 2	
	STRING	Domain Name	
	USINT	Pad [a]	
6	STRING	Host Name	
	USINT	Pad [b]	
7	6 octets	Safety Network Number	0
8	USINT	TTL Value	1
9	STRUCT of	Mcast Config	
	USINT	Alloc control	0
	USINT	Reserved	0
	UINT	Num Mcast	0
	UDINT	Mcast Start Addr	0
10	BOOL	SelectAcd	0
11	STRUCT of	LastConflictDetected	
	USINT	AcdActivity	0
	ARRAY of 6 USINT	RemoteMAC	0
	ARRAY of 28 USINT	ArpPdu	0
12	BOOL	QuickConnect	0
13	UINT	Encapsulation Inactivity Timeout	0
16	UINT	Active TCP Connections	0
17	UDINT	Non-Type 2 Encapsulation Messages per Second	0

[a] Pad octet only included if the Domain Name length is odd.

[b] Pad octet only included if the Host Name length is odd.

Important: Default value shall be inserted for all unsupported attributes that are included in the response.

The lengths of the physical link object path, domain name, and host name are not known before issuing the Get_Attributes_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).

7.5.6.3 Set_Attributes_All request

The instance Set_Attributes_All request shall contain the configuration control attribute, followed by the interface configuration attribute, as specified in Table 65.

Table 65 – Set_Attributes_All request format

Attribute ID	Data Type	Attribute Name	Default Value (if not implemented)
3	DWORD	Configuration Control	
5	STRUCT of	Interface Configuration	
	UDINT	IP Address	
	UDINT	Network Mask	
	UDINT	Gateway Address	
	UDINT	Name Server	
	UDINT	Name Server 2	
	STRING	Domain Name	
	USINT	Pad [a]	

[a] Pad octet only included if the Domain Name length is odd.

When the Configuration Method value in the Configuration Control attribute specified in the request is non-zero, the Interface Configuration attribute value contained in the request shall be ignored.

7.5.7 Class specific services

7.5.7.1 General

The TCP/IP Interface object shall support the class specific services as specified in Table 66.

Table 66 – TCP/IP Interface object class specific services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x4C	N/A	Conditional [a]	Set_Port_Admin_State	Configures the state (open/closed) for the IANA ports that can be controlled.
0x4D	N/A	Conditional [b]	Set_Protocol_Admin_State	Configures the state (enabled/disabled) for the IANA protocols that can be controlled.

[a] Required if instance attribute #14 (IANA Port Admin) is supported and at least one port entry has the Configurable bit set in the Admin Capability member.

[b] Required if instance attribute #15 (IANA Protocol Admin) is supported and at least one protocol entry has the Configurable bit set in the Admin Capability member.

7.5.7.2 Set_Port_Admin_State service

The Set_Port_Admin_State service is used to configure the state (open/closed) for the ports in attribute #14 (IANA Port Admin) that has the Configurable bit set in the Admin Capability member.

If the device requires a reset in order for new configuration to take effect (Reset Required bit in the Admin Capability member set), the device shall set the IANA Port State Change Pending bit (Status attribute bit 8). If the device does not require a reset (Reset Required bit in the Admin

Capability member cleared) in order for new configuration to take effect, the changes shall take effect immediately after sending the response.

If any of the requested port configurations fail, then none of the requested configurations shall be applied and the previous values shall be retained.

The request parameter are defined in Table 67.

Table 67 – Set_Port_Admin_State service request parameters

Name	Data Type	Description of Parameter
Port Count	USINT	Number of ports included in the request
Port Array	ARRAY of STRUCT	Array of ports to configure
Port Number	UINT	IANA Port Number
Protocol	USINT	6 = TCP 17 = UDP
Admin State	BOOL	0 = Close 1 = Open

7.5.7.3 Set Protocol_Admin_State service

The Set_Protocol_Admin_State service is used to configure the state (enabled/disabled) for the protocol in attribute #15 (IANA Protocol Admin) that has the Configurable bit set in the Admin Capability member.

If the device requires a reset in order for new configuration to take effect (Reset Required bit in the Admin Capability member set), the device shall set the IANA Protocol State Change Pending bit (Status attribute bit 9). If the device does not require a reset (Reset Required bit in the Admin Capability member cleared) in order for new configuration to take effect, the changes shall take effect immediately after sending the response.

If any of the requested protocol configurations fail, then none of the requested configurations shall be applied and the previous values shall be retained.

The request parameter are defined in Table 68.

Table 68 – Set_Protocol_Admin_State service request parameters

Name	Data Type	Description of Parameter
Protocol Count	USINT	Number of protocols included in the request
Protocol Array	ARRAY of STRUCT	Array of protocols to configure
Protocol Number	USINT	IANA Port Number
Admin State	BOOL	0 = Disable 1 = Enable

7.5.7.4 Error codes

The class specific services Set_Port_Admin_State and Set_Protocol_Admin_State shall support the error codes shown in Table 69, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

Table 69 – Class specific error codes

General status	Extended status	Error condition
0x20	0x0001	Port Count or Protocol Count too large
0x20	0x0002	Invalid Port Number
0x20	0x0003	Invalid Protocol
0x20	0x0004	Invalid Port Number/Protocol combination
0x20	0x0005	Requested port cannot be closed
0x20	0x0006	Requested protocol cannot be disabled

7.5.8 Behavior

The behavior of the TCP/IP Interface object shall be as illustrated in the State Transition Diagram shown in Figure 21 and Figure 22.

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.

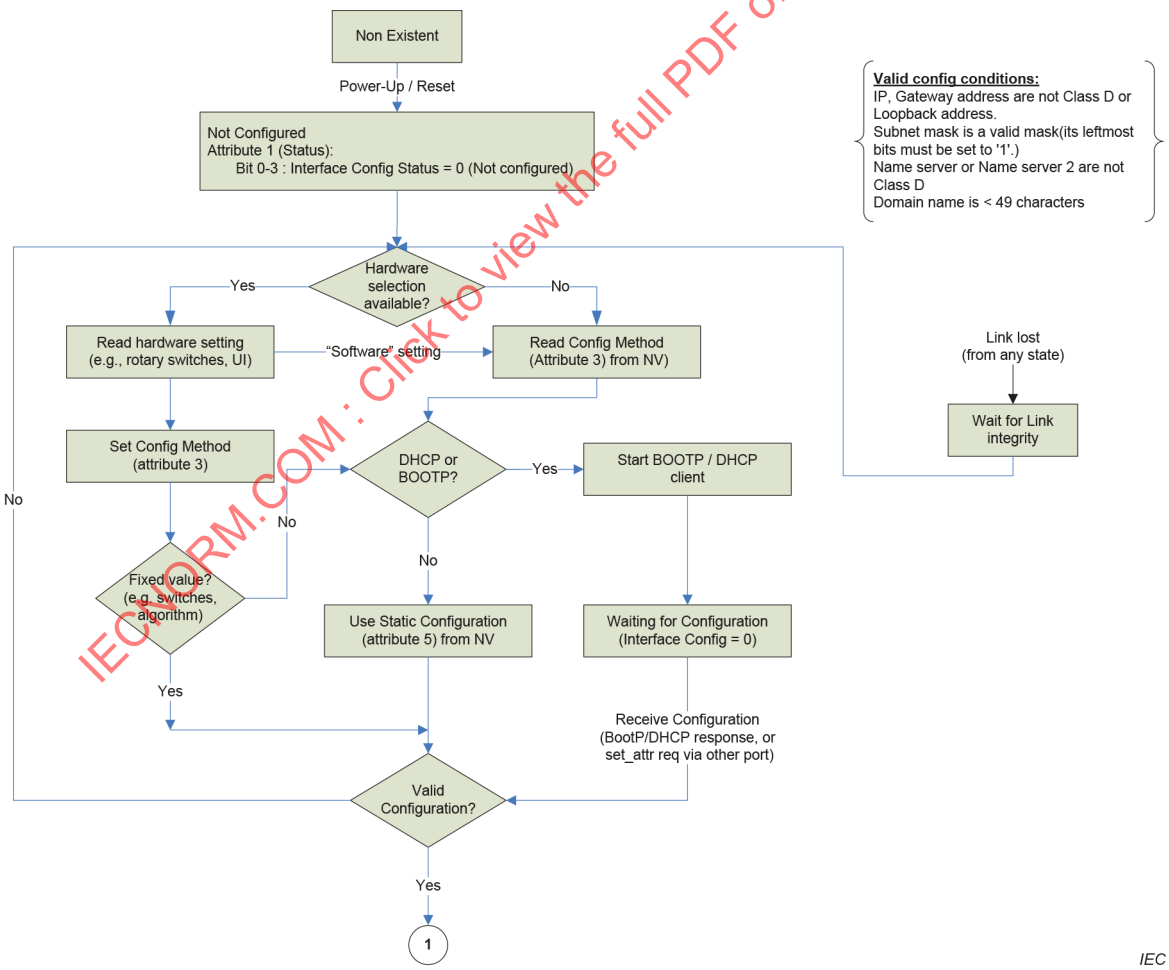


Figure 21 – State transition diagram for TCP/IP Interface object

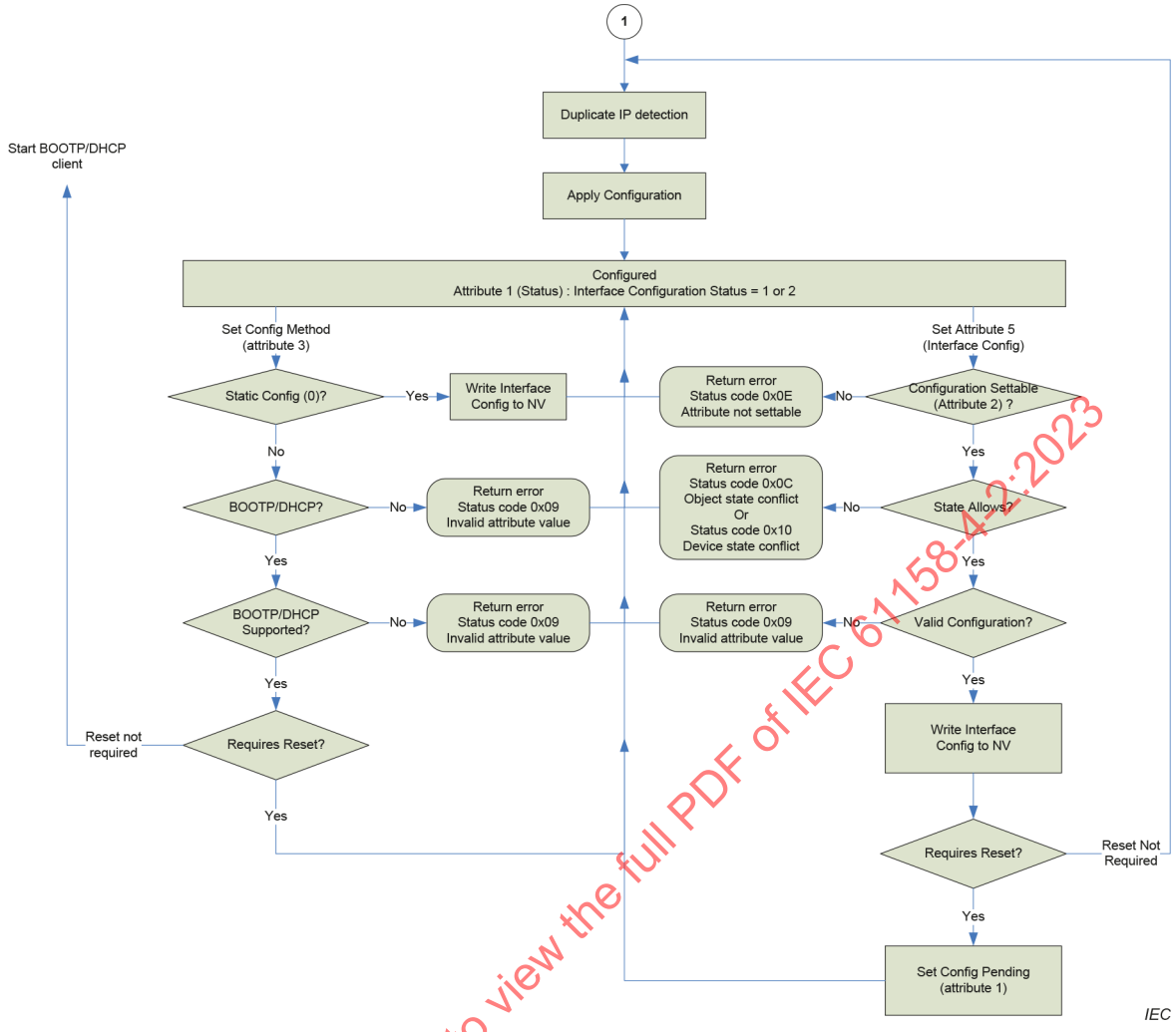


Figure 22 – State transition diagram for TCP/IP Interface object

7.5.9 Address Conflict Detection (ACD)

7.5.9.1 General

7.5.9.1.1 ACD requirements for Type 2 devices

Address conflict detection (ACD) is a mechanism that devices can use to detect and act upon IPv4 address conflicts.

The ACD mechanism specified in this Subclause 7.5.9 conforms to IETF RFC 5227. The requirements specified in IETF RFC 5227 are included by reference except where superseded by normative requirements in 7.5.9. This Subclause 7.5.9 specifies additional requirements for Type 2 devices with respect to the IPv4 ACD mechanism.

The following ACD requirements apply to Type 2 devices:

- ACD is recommended;
- a device implementing ACD shall conform to the ACD mechanism specified in IETF RFC 5227 except where superseded by normative requirements in this Subclause 7.5.9;
- a device executing ACD shall execute the ACD mechanism regardless of the method used by the device for obtaining its IP parameter set;
- a device, executing ACD, and not executing the QuickConnect algorithm shall execute the ACD mechanism before using an IP parameter set;

- devices shall respond to ARP Probes.

7.5.9.1.2 Overview of ACD mechanism in IETF RFC 5227

The IPv4 ACD mechanism described by IETF RFC 5227 involves the following activities:

- initial address probing & conflict detection – before using an IP address the device issues ARP Probes to detect whether the address is in use by another device;
- address announcement – an ARP Request packet is sent after determination that there are no IP address conflicts;
- ongoing conflict detection – ongoing process that is in effect for as long as a device is using an IP Address;
- address defense – procedure used to resolve an address conflict.

7.5.9.2 ACD behavior

The principles of operation for the ACD mechanism are specified in IETF RFC 5227.

In particular, IETF RFC 5227, 2.1, requires that a host implementing this specification shall test to see if the address is already in use when a link-state change signals that an Ethernet cable has been connected.

In addition, this Subclause 7.5.9 further refines the ACD behavior description in the following ways.

– The activities of ACD are grouped into regions, namely:
  - active phase;
  - passive phase:
  - semi-active phase.
– A more complete set of link_up transitions are included in the diagram.
– Both single port and multi-port devices are accommodated.

A device shall implement the ACD mechanism with the behavior specified in Figure 23.

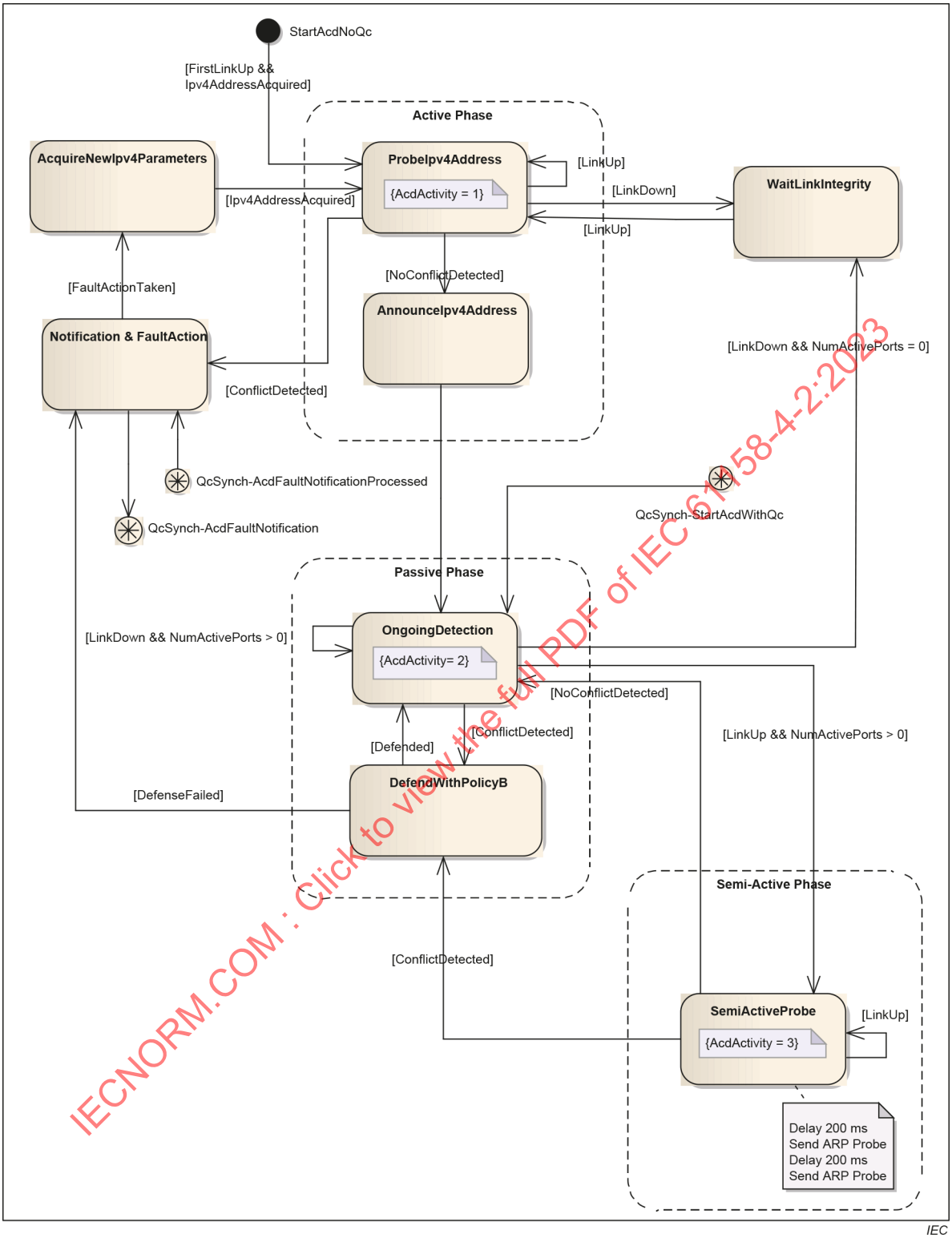


Figure 23 – ACD Behavior

7.5.9.3 ACD details

7.5.9.3.1 IPv4 ACD Timing Constants

IETF RFC 5227, 1.1, specifies a number of timing-related constants. Some of these constants are not optimal for Type 2 applications and networks. In particular, the start-up delays caused by the ARP Probe intervals would be unacceptable in the industrial setting. Adaptation of the specified alternate values is consistent with IETF RFC 5227, 1.3, on Applicability.

Devices that provide ACD shall implement the following alternate values for the parameters defined in IETF RFC 5227, 1.1.

- PROBE_WAIT: 200 ms (initial random delay)
- PROBE_NUM: 4 (number of probe packets)
- PROBE_MIN: 200 ms (minimum delay until repeated probe)
- PROBE_MAX: 200 ms (maximum delay until repeated probe)
- ANNOUNCE_WAIT: 200 ms (delay before announcing)
- ANNOUNCE_INTERVAL: 2 s (delay between announce packets)
- DEFEND_INTERVAL: 2 s (minimum interval between defensive ARPs)

Devices shall comply with the timing in IETF RFC 5227 as modified above within ±10 %. Notice that PROBE_MIN and PROBE_MAX being set to the same value represents the Type 2 recommendation to use a fixed interval between probes rather than the recommendation in IETF RFC 5227, 2.1.1 (Probe Details) that subsequent probes after the initial probe be randomized between PROBE_MIN and PROBE_MAX.

7.5.9.3.2 ProbeIpv4Address

See IETF RFC 5227, 2.1 and 2.1.1.

In addition to the requirements specified in IETF RFC 5227, only ARP probes sent to the Ethernet broadcast address shall be considered for conflict detection. Directed ARP messages (sent to the device's Ethernet MAC address) with Sender IP Address of 0.0.0.0 shall not be treated as a conflict.

NOTE This requirement does not conflict with IETF RFC 5227, which defines ARP probes as being broadcast on the local link.

7.5.9.3.3 AnnounceIpv4Address

See IETF RFC 5227, 2.3.

When the device has sent its first ARP Announcement it shall transition to OngoingDetection.

NOTE The OngoingDetection phase occurs concurrent with completing the AnnounceIPv4Address phase.

7.5.9.3.4 WaitLinkIntegrity

The WaitLinkIntegrity activity waits for the occurrence of a LinkUp event from an Ethernet port.

This activity is initiated in two scenarios:

- one is when a single port device experiences a LinkDown event;
- the other is when a multi-port device experiences a LinkDown event and all of its other Ethernet ports are also down.

When a LinkUp event occurs the ProbeIpv4Address activity is then initiated.

When a device detects that Ethernet link integrity has been lost and then regained (e.g., the network cable has been removed and replaced), the device shall restart the initial ProbeIpv4Address activity.

7.5.9.3.5 Notification & Fault Action

See IETF RFC 5227, Clause 1.

In addition to IETF RFC 5227 behavior, when an address conflict is detected, Type 2 devices shall take the following fault actions:

- set the device state to Major Recoverable Fault (Module Status indicator flashing red);
- set the Network Status indicator to solid red.

There are 2 synchronization transitions between the ACD Notification & FaultAction activity and the QuickConnect behavior diagram.

The QcSynch-AcdFaultNotification is a synchronization transition to the QuickConnect behavior diagram that occurs either when the ACD mechanism has a ConflictDetected fault from the ProbeIpAddress activity or a DefenseFailed fault from the DefendWithPolicyB activity.

The QcSynch-AcdFaultNotificationProcessed is a synchronization transition from the QuickConnect behavior diagram to the Notification & FaultAction activity that occurs after QuickConnect has processed the previous AcdFaultNotification.

7.5.9.3.6 AcquireNewIpv4Parameters

See IETF RFC 5227, Clause 1.

After notification of the detection of an IPv4 Address conflict, the device may be designed to obtain or use an alternate IPv4 address. The design of this method and its resulting selection of IPv4 Parameters are vendor specific.

If a LinkDown occurs on the last active port while in the AcquireNewIpv4Parameter activity, then the ACD process in Figure 23 (ACD Behavior) shall terminate. The ACD process shall be restarted after at least one link has been reestablished and a valid IPv4 configuration has been acquired according to Figure 21 and Figure 22 showing the behavior of the TCP/IP Interface object.

Notice also that Type 2 Ethernet devices shall limit the rate at which they probe for new candidate addresses using MAX_CONFLICTS and RATE_LIMIT_INTERVAL as defined in IETF RFC 5227, 2.1.1.

7.5.9.3.7 OngoingDetection

See IETF RFC 5227, 2.4.

IETF RFC 5227, 2.4 states that "Address Conflict Detection is not limited to only the time of initial interface configuration, when a host is sending ARP Probes. Address Conflict Detection is an ongoing process that is in effect for as long as a host is using an address."

After an IP address has successfully been probed and put into use, a device shall perform ongoing conflict detection and defense according to IETF RFC 5227.

The QcSynch-StartAcdWithQc is a synchronization transition with the QuickConnect behavior diagram. If QuickConnect is enabled in the device, the activity for ACD begins at this point.

The ACD mechanism as specified in IETF RFC 5227, 2.1 states that a host shall not perform this check periodically as a matter of course, indicating that periodic ARP Probes are not required to be sent as part of ongoing detection. However it has been observed that periodic ARP Probes allow the module to detect conflicts with devices that had possibly not been connected to the network at the time of initial probing, or when switches have dropped the initial ARP Probes due to initial forwarding delay. Therefore, Type 2 devices that provide the ACD algorithm should issue periodic ARP Probes during ongoing detection.

Type 2 devices that support periodic ARP Probes during ongoing detection shall use a transmission delay in the range of ONGOING_PROBE_MIN and ONGOING_PROBE_MAX using the values below:

- ONGOING_PROBE_MIN: 90 s (minimum time interval for ongoing probes);
- ONGOING_PROBE_MAX: 150 s (maximum time interval for ongoing probes).

These values are not defined in IETF RFC 5227. Devices shall not exceed this specified range by more than 10 % on either end.

It is recommended that the initial transmission delay for the first periodic ARP Probe has a pseudo-random value within the range of ONGOING_PROBE_MIN and ONGOING_PROBE_MAX ("randomized" using the Ethernet MAC address, as shown for example in 7.5.9.3.9). The transmission delay for subsequent periodic ARP Probes need not be randomized, and may, for example, select the center value (120 s).

7.5.9.3.8 DefendWithPolicyB

See IETF RFC 5227, 2.4.

If an address conflict is detected, the device shall defend its address per alternative (b) in IETF RFC 5227, 2.4.

If a conflict persists (as defined by IETF RFC 5227) the device shall immediately cease using this address and execute the notification and fault actions specified in 7.5.9.3.5.

7.5.9.3.9 SemiActiveProbe

The SemiActiveProbe activity is initiated when a multi-port device receives a LinkUp event while other ports are still active.

The SemiActiveProbe activity is a modified probing activity in which only two ARP probes are sent using probe delays of 200 ms.

The SemiActiveProbe activity is defined to reduce the amount of ARP broadcast traffic that will be initiated from LinkUp activity on multi-port devices.

7.5.9.3.10 Example pseudo-random delay algorithm (informative)

NOTE The following Xorshift Random Number Generator algorithm is based on the work of George Marsaglia from Florida State University.

This Random Number Generator (RNG) is used to produce an initial pseudo-random delay before sending out the first periodic ARP probe based on the PROBE_WAIT value, and may also be used for determining the initial transmission delay the periodic ARP probes using the ONGOING_PROBE_MIN and ONGOING_PROBE_MAX values.

IETF RFC 5227 requires that the device wait for a random time interval in the range zero to PROBE_WAIT. Assuming a PROBE_WAIT of 200 ms, then the RNG algorithm can be used to select an initial delay in the range of 0 s to 200 ms using the following formula:

Initial_Probe_Delay = PROBE_WAIT × R256 / 256

R256 is a random number in the range (0 .. 255) calculated using the algorithm below.

The RNG algorithm uses an IEEE Std 802.3-2018 MAC address, eg 12-34-56-78-9A-BC, written here in canonical notation. Canonical notation represents the order in which the octets of the MAC address are transmitted, left to right, on the wire; i.e. 0x12 sent first, 0x34 sent next, and

so on. The MAC address is mapped to an array, EnetAddr[], of 6 octets as follows, using the example MAC address.

EnetAddr[0] = 0x12;
EnetAddr[1] = 0x34;
EnetAddr[2] = 0x56;
EnetAddr[3] = 0x78;
EnetAddr[4] = 0x9A;
EnetAddr[5] = 0xBC;

R256 is then calculated using the following Xorshift RNG.

R256 = EnetAddr[0];
R256 = (R256 << 1) ^ EnetAddr[1];
R256 = (R256 << 1) ^ EnetAddr[2];
R256 = (R256 << 1) ^ EnetAddr[3];
R256 = (R256 << 1) ^ EnetAddr[4];
R256 = (R256 << 1) ^ EnetAddr[5];
R256 = R256 % 256;

Where << is the left shift operator, ^ is the exclusive OR operator, and % is the modulus operator.

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/IEEE 8802-3 port. Each device shall support exactly one instance of the Ethernet Link object for each Ethernet port on the module. Devices may use an Ethernet Link object instance for an internally accessible interface, such as an internal port for an embedded switch.

7.6.2 Revision history

The revision history of the Ethernet Link object is described in Table 70.

Table 70 – Ethernet link object revision history

Class revision	Description of changes
01	Initial Release
02	Release for IEC 61158.
03	Add new instance attributes 7-10 providing support for multiple port Ethernet devices
04	Add 1 Gbit/s Forced Speed Setting behavior; add Instance Attribute 11, Interface Capability; add Instance Attributes 12 and 13 (HC counters)

7.6.3 Class attributes

The Ethernet Link object shall support the class attributes as specified in Table 71.

Table 71 – Ethernet link object class attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	NV	Revision	UINT	Revision of this object	The current value shall be 4.
2	Conditional [a]	Get	NV	Max Instance	UINT	Maximum instance number of an object currently created in this class level of the device	The largest instance number of a created object at this class hierarchy level
3	Conditional [a]	Get	NV	Number of Instances	UINT	Number of object instances currently created at this class level of the device	The number of object instances at this class hierarchy level
4 to 7	These class attributes are optional and are described in IEC 61158-5-2.						

[a] Required if the number of instances is greater than 1.

7.6.4 Instance attributes

7.6.4.1 General

The Ethernet Link object shall support the instance attributes as specified in Table 72.

Table 72 – Ethernet link object instance attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	V	Interface speed	UDINT	Interface speed currently in use	Speed in Mbit/s (e.g. 0, 10, 100, 1 000). See 7.6.4.2
2	Required	Get	V	Interface flags	DWORD	Interface status flags	Bit map of interface flags. See 7.6.4.3
3	Required	Get	NV	Physical address	USINT[6]	MAC layer address	See 7.6.4.4
4	Conditional [a]	Get	V	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	

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
				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	
				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	
5	Conditional [b]	Get	V	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	

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
				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 per-mitted DLPDU size	
				MAC receive errors	UDINT	DLPDUs for which reception on an interface fails due to an internal MAC sublayer receive error	
6	Optional	Get/Set	NV	Interface control	STRUCT of	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)
7	Optional	Get	NV	Interface Type	USINT	Type of interface, e.g. twisted pair, fiber, internal	See 7.6.4.8
8	Optional	Get	V	Interface State	USINT	Current state of the interface, e.g. operational, disabled	See 7.6.4.9
9	Optional	Set	NV	Admin State	USINT	Administrative state: enable, disable	See 7.6.4.10
10	Conditional [c]	Get	NV	Interface Label	SHORT_STRING_	Human readable identification	See 7.6.4.11

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
11	Required	Get	NV	Interface Capability	STRUCT of variable size	Indication of capabilities of the interface	See 7.6.4.12
				Capability Bits	DWORD	Interface capabilities, other than speed/duplex	Bit map
				Speed/Duplex Options	STRUCT of variable size	Indicates speed/duplex pairs supported in the Interface Control attribute	
					USINT	Speed/Duplex Array Count	Number of elements
					ARRAY of STRUCT of	Speed/Duplex Array	
					UINT	Interface Speed	Semantics are the same as the Forced Interface Speed in the Interface Control attribute: speed in Mbit/s.
					USINT	Interface Duplex Mode	0=half duplex 1=full duplex 2 to 255 = Reserved
12	Conditional [d]	Get	V	HC Interface Counters	STRUCT of 32 octets	High Capacity Interface Counters	See 7.6.4.13
				HCInOctets	ULINT	The total number of octets received on the interface. This counter is a 64-bit version of In Octets.	
				HCInUcastPkts	ULINT	Unicast packets received on the interface. This counter is a 64-bit version of In Ucast Packets	
				HCInMulticastPkts	ULINT	Multicast packets received on the interface.	
				HCInBroadcastPkts	ULINT	Broadcast packets received on the interface.	
				HCOutOctets	ULINT	Octets sent on the interface. This counter is a 64-bit version of Out Octets.	
				HCOutUcastPkts	ULINT	Unicast packets sent on the interface. This counter is a 64-bit version of Out Ucast Packets	

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
				HCOutMulticastPkts	ULINT	Multicast packets sent on the interface.	
				HCOutBroadcastPkts	ULINT	Broadcast packets sent on the interface.	
13	Conditional [d]	Get	V	HC Media Counters	STRUCT of 24 octets	High Capacity Media Counters	See 7.6.4.13
				HCStatsAlignmentErrors	ULINT	Frames received that are not an integral number of octets in length and do not pass the FCS check. This counter is a 64-bit version of Alignment Errors.	
				HCStatsFCSErrors	ULINT	Frames received that are an integral number of octets in length but do not pass the FCS check. This counter is a 64-bit version of FCS Errors.	
				HCStatsInternalMacTransmitErrors	ULINT	Frames for which transmission fails due to an internal MAC sublayer transmit error. This counter is a 64-bit version of MAC Transmit Errors.	
				HCStatsFrameTooLongs	ULINT	Frames received that exceed the maximum permitted frame size. This counter is a 64-bit version of Frame Too Long Errors.	
				HCStatsSymbolErrors	ULINT	Number of times there was an invalid data symbol on the media when a valid carrier was present.	
				HCStatsInternalMacReceiveErrors	ULINT	Frames for which reception on an interface fails due to an internal MAC sublayer receive error. This counter is a 64-bit version of MAC Receive Errors.	

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
14	Conditional [e]	Get	V	Ethernet Errors	UDINT	Sum of certain error counts that are part of attributes 4, 5 and 13	See 7.6.4.14
15	Optional	Get	V	Link Down Counter	UDINT	Counts the number of times a Link Down transition event was detected on this port	See 7.6.4.15

[a] The Interface counters attribute is required if the HC Interface Counters attribute or Media Counters attribute is implemented, otherwise highly recommended.

[b] Required if the device supports DLR or if the HC Media Counters attribute is implemented, otherwise highly recommended.

[c] Required if number of instances is greater than 1.

[d] Required for interfaces that support 1Gbit/s speeds and higher, otherwise highly recommended.

[e] Optional, but recommended when the media and interface counters are implemented, otherwise not permitted.

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 73.

Table 73 – 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 could 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.

Bit	Name	Definition
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 10 Mbit/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	The Manual Setting Requires Reset bit is the same as the identically-named bit in the Interface Capability attribute (#11). This bit shall be duplicated in both attributes in order to retain backwards compatibility with previous object revisions.
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/IEEE 8802-3 requirements.

Devices with multiple ports but a single MAC interface (e.g., a device with a embedded switch technology) may use the same value for this attribute in each instance of the Ethernet Link object. The general requirement is that the value of this attribute shall be the MAC address used for packets to and from the device's own MAC interface over this physical port.

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 IETF RFC 1213. The interface counters is a conditional attribute; it shall be implemented if the media counters attribute is implemented.

Multi-port devices with a single MAC interface (e.g., device with an embedded switch) shall maintain counter values in one of three ways.

- In each instance, count the MAC frames sent/received by the device itself over the port represented by that instance (i.e., each physical interface counts the MAC frames sent/received over that interface). This is the preferred solution.
- Use counter values of 0 for those instances that correspond to the external switch ports; count MAC frames in the instance that corresponds to the internal device interface.
- Use the same counter values for all instances, counting MAC frames sent/received by the device itself.

7.6.4.6 Media counters

The media counters attribute contains counters specific to Ethernet media. These counters shall be as defined by IETF RFC 1643. If this attribute is implemented the interface counters attribute shall also be implemented. Instances that refer to internal interfaces may set the values of the Interface Counters to 0.

It is possible that some underlying hardware or system implementations do not provide all of the Media Counters. In the case of fiber media, some of the counters do not apply (e.g., collision counters). Devices shall use values of 0 for counters that are not 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 7.6.4.7.2 to 7.6.4.7.3.

7.6.4.7.2 Control bits

The control bits attribute is specified in Table 74.

Table 74 – Control bits

Bit	Name	Definition
0	Auto-negotiate	0 indicates ISO/IEC/IEEE 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 a status code 0x09 (Invalid Attribute Value). If auto-negotiation is enabled, attempting to set the forced duplex mode bits shall result in status 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).

For Gigabit Ethernet speeds (1 Gbit/s and above), the standard auto negotiation procedure is mandatory according to IEEE Std 802.3-2018. Setting the Auto-negotiate bit to 0 and Forced Interface Speed to 1 000 or above shall cause the interface to advertise only the specified speed and only the specified Forced Duplex Mode during link negotiation (according to IEEE Std 802.3-2018).

Interfaces not supporting the requested speed shall return status code 0x09 (Invalid Attribute Value).If auto-negotiation is enabled, attempting to set the Forced Interface Speed shall result in status code 0x0C (Object State Conflict).

7.6.4.8 Interface type

The Interface Type attribute indicates the type of the physical interface. Table 75 specifies the Interface Type values. This attribute shall be stored in non-volatile memory.

Table 75 – Interface type

Value	Type of interface
0	Unknown interface type
1	The interface is internal to the device, for example, in the case of an embedded switch
2	Twisted-pair (e.g., 10Base-T, 100Base-TX, 1000Base-T)
3	Optical fiber (e.g., 100Base-FX)
4-255	Reserved

7.6.4.9 Interface state

The Interface State attribute indicates the current operational state of the interface. Table 76 specifies the Interface State values. This attribute shall be stored in volatile memory.

Table 76 – Interface state

Value	Interface state
0	Unknown interface state
1	The interface is enabled and is ready to send and receive data
2	The interface is disabled
3	The interface is testing
4-255	Reserved

7.6.4.10 Admin state

The Admin State attribute allows administrative setting of the interface state. Table 77 specifies the Admin State values. This attribute shall be stored in non-volatile memory.

Table 77 – Admin state

Value	Admin state
0	Reserved
1	Enable the interface
2	Disable the interface.
3-255	Reserved

Devices whose only communications port is a Type 2 Ethernet port with a single Ethernet Link instance shall return general status code 0x09 (Invalid Attribute Value) if a request to disable its interface is received. Devices with multiple ports (any combination of multiple Ethernet Link instances and/or other communications ports) shall return general status code 0x10 (Device State Conflict) if performing a disable request for an interface would result in all of the device's communication ports becoming disabled.

7.6.4.11 Interface label

The Interface Label attribute is a text string that describes the interface. The content of the string is vendor specific. For internal interfaces the text string should include "internal" somewhere in the string. The maximum number of characters in this string is 64. This attribute shall be stored in non-volatile memory.

7.6.4.12 Interface Capability

The Interface Capability attribute shall indicate the set of capabilities for the interface. The attribute is a structure with two main elements: Capability Bits and Speed/Duplex Options.

Capability Bits contains an array of bits that indicate whether the interface supports capabilities such as auto-negotiation and auto-MDIX. Table 78 specifies the capability bits.

Table 78 – Capability Bits

Bit(s):	Called:	Definition
0	Manual Setting Requires Reset	Indicates whether or not the device requires a reset to apply changes made to the Interface Control attribute (#6). 0 = Indicates that the device automatically applies changes made to the Interface Control attribute (#6) and, therefore, does not require a reset in order for changes to take effect. This is the value this bit shall have when the Interface Control attribute (#6) is not implemented. 1 = Indicates that the device does not automatically apply changes made to the Interface Control attribute (#6) and, therefore, will require a reset in order for changes to take effect. This bit shall also be replicated in the Interface Flags attribute (#2) in order to retain backwards compatibility with previous object revisions.
1	Auto-negotiate	0 = Indicates that the interface does not support link auto-negotiation 1 = Indicates that the interface supports link auto-negotiation
2	Auto-MDIX	0 = Indicates that the interface does not support auto MDIX operation 1 = Indicates that the interface supports auto MDIX operation
3	Manual Speed/Duplex	0 = Indicates that the interface does not support manual setting of speed/duplex. The Interface Control attribute (#6) shall not be supported. 1 = Indicates that the interface supports manual setting of speed/duplex via the Interface Control attribute (#6)
4 – 31	Reserved	Shall be set to 0

The Speed/Duplex Options element holds an array that indicates the speed/duplex pairs that can be set via the Interface Control instance attribute (#6). One speed/duplex pair (e.g. 10 Mbit/s-half duplex, 100 Mbit/s-full duplex) shall be returned for each combination supported by the interface.

7.6.4.13 High Capacity (HC) Counters

The two HC counter attributes (attributes 12 and 13) contain 64-bit versions of the equivalent 32-bit counters (attributes 4 and 5). The 64-bit counters have the same basic semantics as their 32-bit counterparts, extended to 64 bits.

The HC Interface Counters attribute contains 64-bit versions of the Interface Counters. Counters in this attribute are in conformance with the Interfaces Group MIB defined by IETF RFC 2863.

NOTE In the 32-bit Interface Counters, the counters for multicast and broadcast packets are combined as non-unicast packets.

The HC Media Counters attribute contains 64-bit versions of the Media Counters. Counters in this attribute are in conformance with the Ethernet-Like MIB defined by IETF RFC 3635.

For interfaces that have the capability to operate at 1 Gbit/s or faster, 64-bit counters shall be implemented and report accurate values when the interface is operating at any of its supported speeds. When 64-bit counters are in use, the 32-bit counters shall still be available and shall report the low 32 bits of the associated 64-bit counter.

7.6.4.14 Ethernet Errors

This attribute is the sum of the values from the following specific members of the Ethernet Link object instance attributes:

- Interface Counters, Attribute 4: In Discards, In Errors, Out Discards, Out Errors;
- Media Counters, Attribute 5: all members;
- HC Media Counters, Attribute 13 (if supported): HCStatsSymbolErrors.

NOTE All other members in this attribute are counted by members in Media Counters, Attribute 5.

A Get_and_Clear service to any of the underlying attributes listed above will result in the Ethernet Errors attribute value decrementing accordingly. In order to clear the Ethernet Errors attribute the underlying attributes should be cleared via the Get_and_Clear service, or alternatively cleared as a function of the client application that reads the counters.

7.6.4.15 Link Down Counter

The Link Down Counter attribute shall increment when a transition from Link_Up to Link Down event is detected on this port. The device may also include transitions caused by internal events, including Admin State changing to Disabled, Local Hardware Faults, or entering Testing mode. Not all devices will have the ability to detect each and every Link Down that occurs, especially when they occur rapidly. This is affected by different data rates, whether the link is set for fixed speed/duplex or auto-negotiation, differences in PHY parts and whether recognition of Link Down in the device is interrupt driven or polled.

7.6.5 Diagnostic connection points

One diagnostic connection point is defined for the Ethernet Link object class at the instance level. This connection point is required if the Standard Network Diagnostic Assembly is implemented (see IEC 61158-6-2, 4.1.8.4.14).

NOTE See IEC 61158-5-2, 6.1.6 for further information about Diagnostic Connection Points.

The format of the Ethernet Link diagnostic connection point is as specified in Table 79.

Table 79 – Ethernet Link connection point 1, Standard Network Diagnostics

Attribute ID	Attribute name	Bit	Data type	Attribute size	Size of structure
Part of attribute 2 [a]	Link Status	0	BOOL	1 bit	16 octets
	Half/Full Duplex	1	BOOL	1 bit	
	Negotiation Status	2-4	BOOL	3 bits	
	Permanently reserved, shall be zero	5	BOOL	1 bit	
	Local Hardware Fault	6	BOOL	1 bit	
N/A	25 bits pad, shall be zeros			25 bits	
1	Interface Speed		UDINT	4 octets	
15	Link Down Counter		UDINT	4 octets	
14	Ethernet Errors		UDINT	4 octets	

[a] The value returned in field Size of Member Data for the Get_Connection_Point_Member_List service response shall be 7 bits.

7.6.6 Common services

7.6.6.1 General

The Ethernet Link object shall support the common services as specified in Table 80.

Table 80 – Ethernet Link object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Optional	Get_Attributes_All	Returns a predefined listing of this objects attributes (see the Get_Attributes_All response definition in 7.6.6.2)
0x0E	Conditional	Required	Get_Attribute_Single[a]	Returns the contents of the specified attribute or connection point
0x10	N/A	Conditional	Set_Attribute_Single	Modifies a single attribute
0x1D	N/A	Conditional[b]	Get_Connection_Point_Member_List	Used to get member paths of a Connection Point structure

[a] This service shall support the use of a connection point in the service path when Diagnostic Connection Point(s) are implemented (see 7.6.5).

[b] This service is required if Diagnostic Connection Point(s) are implemented.

The Get_Attribute_Single service shall be implemented for the class if any class attribute is implemented.

The Set_Attribute_Single service shall be implemented if the interface control or admin state attributes are implemented.

7.6.6.2 Get_Attributes_All response

At the class level, the Get_Attributes_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

At the instance level, the order of the attributes returned in the Get_Attributes_All response shall be as specified in Table 81.

Table 81 – Get_Attributes_All response format

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	UDINT	Interface Speed	
2	DWORD	Interface Flags	
3	ARRAY of 6 USINT	Physical Address	
4	STRUCT of 11 UDINT	Interface Counters	0
5	STRUCT of 12 UDINT	Media Counters	0
6	STRUCT of	Interface Control [a]	
	WORD	Control Bits	0
	UINT	Forced Interface Speed	0
7	USINT	Interface Type	0
8	USINT	Interface State	0
9	USINT	Admin State	0
10	SHORT_STRING	Interface Label	Zero-length string
11	STRUCT of	Interface Capability	
	DWORD	Capability Bits	
	STRUCT of	Speed/Duplex Options	
	USINT	Speed/Duplex Array Count	
	ARRAY of:	Speed/Duplex Array	
	STRUCT of	Speed/Duplex Pair	
	UINT	Interface Speed	
	USINT	Interface Duplex Mode	
12	STRUCT of 8 ULINT	HC Interface Counters	0
13	STRUCT of 6 ULINT	HC Media Counters	0
14	UDINT	Ethernet Errors	0
15	UDINT	Link Down Counter	0

[a] The default value for this attribute is an otherwise invalid combination (since Auto Negotiation is disabled in Control Bits, but Forced Interface Speed is zero) and can therefore be used to determine that the attribute is not supported.

Important: The default value shall be inserted for all unsupported attributes that are included in the response.

The lengths of the Interface Label and Speed/Duplex Options are not known before issuing the Get_Attributes_All service request. Implementers shall be prepared to accept a response containing the maximum size of the Interface Label (65 USINT's) and an Interface Capability attribute with at least 10 elements in its Speed/Duplex Array.

7.6.7 Class specific services

7.6.7.1 General

The Ethernet Link object shall support the class specific services as specified in Table 82.

Table 82 – 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, Media Counters, HC Interface Counters or HC Media Counters)

The Get_and_Clear service is required if one or more of the attributes supported by this service (see 7.6.7.2) are implemented, and is not allowed otherwise.

7.6.7.2 Get_and_Clear service

The Get_and_Clear service is a class specific service. 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.

This service shall be supported by the following attributes (if the attribute itself is supported): Interface Counters (#4), Media Counters (#5), HC Interface Counters (#12), HC Media Counters (#13), and Link Down Counter (#15). This service shall not be supported by any other open attributes, but may be supported by vendor-specific attributes.

When this service is directed at either of the Interface Counters (#4) or HC Interface Counters (#12) attributes and both are supported, then both attributes shall be cleared by this service. When this service is directed at either of the Media Counters (#5) or HC Media Counters (#13) attributes and both are supported, then both attributes shall be cleared by this service.

7.6.8 Behavior

The behavior of the TCP/IP Interface object shall be as illustrated in the State Transition Diagram shown in Figure 24.

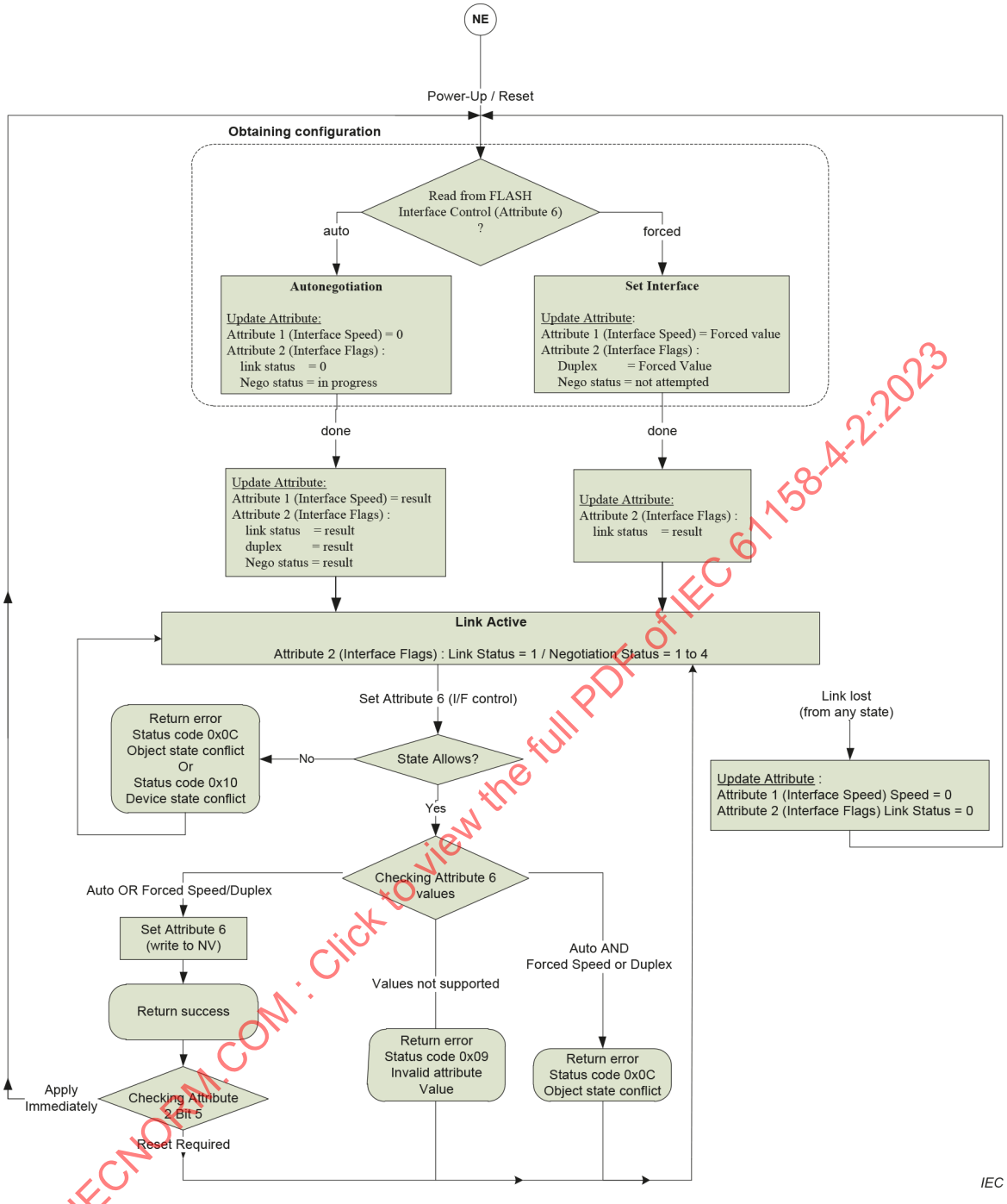


Figure 24 – State transition diagram for Ethernet Link object

7.7 DeviceNet™ [4] 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.

---

4 DeviceNet™ is a trade name of ODVA, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance with a related profile does not require use of the trade name. Use of the trade name requires permission of the trade name holder.

A device may contain more than one node.

7.7.2 Revision history

The revision history of the DeviceNet object is described in Table 83.

Table 83 – 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 84.

Table 84 – DeviceNet object class attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	NV	Revision	UINT	revision of this object	2 (revision 2)
2 to 7	These class attributes are optional and are described in IEC 61158-5-2.						

7.7.4 Instance attributes

7.7.4.1 General

The DeviceNet object shall support the instance attributes as specified in Table 85.

Table 85 – DeviceNet object instance attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Optional	Get/Set	NV	MAC ID	USINT	node address	see 7.7.4.2
2	Optional	Get/Set	NV	Bit Rate	USINT	bit rate	see 7.7.4.3
3	Optional	Get/Set	NV	BOI	BOOL	bus-off interrupt	see 7.7.4.4
4	Optional	Get/Set	V	Bus-Off Counter	USINT	number of times CAN went to the bus-off state	see 7.7.4.5
5	Conditional[c]	Get	V	Allocation Information	STRUCT of 2 octets		see 7.7.4.6
				Allocation choice	SWORD		see 7.7.4.6.2
				Controller's MAC ID	USINT	MAC ID of Controller (from allocate)	see 7.7.4.6.3
6	Conditional[a]	Get	V	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

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
7	Conditional[b]	Get	V	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
8	Conditional[a]	Get	V	MAC ID switch value	USINT	Actual value of node address switch(es)	0 to 99
9	Conditional[b]	Get	V	Bit rate switch value	USINT	Actual value of bit rate switch(es)	0 to 9
10	Optional	Get/Set	NV	QuickConnect	BOOL	Enable/Disable of QuickConnect feature	0 = disable (default) 1 = enable
11	Conditional[d]			Safety Network Number	SWORD[6]		See IEC 61784-3-2
12	Optional	Get	V	Diagnostic counters	STRUCT of 34 octets	List of ISO 11898-1 diagnostic counters	See 7.7.4.7
				Diagnostic counters descriptor	WORD	Indicates which diagnostic counters are supported	
				Arbitration loss count	UINT	See ISO 11898-1	Range = 0 to 65 535 Default = 0
				Overload count	UINT	See ISO 11898-1	
				Bit error count	UINT	See ISO 11898-1	
				Stuff error count	UINT	See ISO 11898-1	
				Ack error count	UINT	See ISO 11898-1	
				Form error count	UINT	See ISO 11898-1	
				CRC error count	UINT	See ISO 11898-1	
				Rx message loss count	UINT	ISO 11898-1 subsystem has detected a lost receive message	
				Warning error count	UINT	See ISO 11898-1	
				Rx error counter	UINT	Receive error counter (see ISO 11898-1)	Range = 0 to 256
				Tx error counter	UINT	Transmit error counter (see ISO 11898-1)	Range = 0 to 256
					UINT[5]	Reserved	Default = 0

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
13	Conditional[e]	Get	V	Active Node Table	BOOL[64]	Identifies which nodes are online on the local network, based on Node Address	One bit for each node number. The node number is the index into the bit array. 0 = Inactive 1 = Active

[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 is required if the Predefined Controller/Device Connection Set is supported.

[d] This attribute is required for Type 2 safety devices. Non-safety devices shall not implement this attribute.

[e] This attribute is required if the device supports routing between two or more Type 2 ports.

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 Communication Fault 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 scenario applies 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 Link Access State Machine defined in IEC 62026-3:2014, 5.4.

7.7.4.3 Bit Rate

The Bit Rate attribute indicates the selected bit rate. Values are specified in Table 86.

Table 86 – 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 scenario applies 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 87 – 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 11898-1 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 Communication Fault state (see IEC 62026-3:2014, 5.4).

When the BOI attribute is TRUE (set to one) and a CAN chip bus-off event is detected, it could be possible to return the CAN chip to its normal operating mode and continue communicating based on the Link Access State Machine presented in IEC 62026-3:2014, 5.4.

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 affected 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 Link Access State Machine.

As indicated in Table 85, 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 87.

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 is 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 Controller/Device Connection Set. Its support is required if the Predefined Controller/Device Connection Set is supported. It indicates whether or not the Predefined Controller/Device Connection Set defined in IEC 62026-3:2014, 5.5 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.

NOTE The Predefined Controller/Device Connection Set is called Predefined Master/Slave Connection Set in IEC 62026-3:2014.

This attribute is modified when a successful response associated with an Allocate_Controller/Device_Connection_Set service (defined later in 7.7.6) is generated.

7.7.4.6.2 Allocation choice

The Allocation choice indicates which of the Predefined Controller/Device Connections are active (in the Configuring, or Established state). Its format is specified in IEC 62026-3:2014, 5.5.

The Allocation choice is initialized to 00 at device power–up or reset.

7.7.4.6.3 Controller's MAC ID

The Controller's MAC ID contains the MAC ID of the device that has allocated the Predefined Controller/Device Connection Set via the Allocate_Controller/Device_Connection_Set service. This contains the Allocator's MAC ID field copied from the Allocate_Controller/Device_Connection_Set request.

The range of values is 0 to 63 and 255 decimal. A value in the range 0 to 63 indicates that the Predefined Controller/Device Connection Set is currently allocated and denotes the MAC ID of the device that performed the allocation. The value 255 means the Predefined Controller/Device Connection set has not been allocated.

The Controller'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 88 specifies the diagnostic counter bit assignment.

Table 88 – 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
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 fore 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 ISO 11898-1, 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 62026-3:2014, 5.4).

7.7.4.8 Active Node Table

The Active Node Table attribute reports the activity state of each node on the network, based on node numbers. Each bit in the array represents a node on the local network, with the bit position matching the node address represented (i.e. Bit 0 = Node Address 0, Bit 1 = Node Address 1). When the bit is set (TRUE) the node is active on the link. When the bit is cleared (FALSE) the node is inactive on the link.

The bit is set (node is indicated as active) when the node represented by the bit is in either the On-Line, Sending Duplicate MAC ID Check Request Message, or Waiting for Duplicate MAC ID Check Message state. The setting of the bit shall occur within 1 s of a previously inactive node transmitting on the link. Also, the device containing the Active Node Table shall set all bits when transitioning to the On-Line state.

The bit is cleared (node is indicated as inactive) when the node represented by the bit is in either the Non-Existent or Communication Fault state. The clearing of the bit shall occur within 20 s of the previously active node leaving the link (or, if the node is not present on the network, within 20 s after the device containing the Active Node Table transitions to the On-Line state). Also, the device containing the Active Node Table shall clear all bits when transitioning out of the On-Line state.

A router shall not attempt to route an unconnected message request to a node on the local network when the Active Node Table bit for that node indicates inactive (bit is cleared). Instead, the router shall immediately return a General Error status of 0x01 and an Extended Error status of 0x0204.

7.7.5 Common services

7.7.5.1 General

The DeviceNet object shall support the common services as specified in Table 89.

Table 89 – 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
0x0E	Required	Conditional[b]	Get_Attribute_Single	Returns the contents of the specified attribute
0x10	N/A	Conditional[c]	Set_Attribute_Single	Modifies a single attribute

[a] This service is required when this DeviceNet object instance is addressable through some other Type 2 access mechanism other than the subnet interface controlled by this DeviceNet object instance.

[b] The Get_Attribute_Single service is Required if any Instance attributes are implemented.

[c] The Set_Attribute_Single service is Required if any implemented Instance attributes are settable.

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 90.

Table 90 – Reset service parameter

Name	Type	Description of request parameters	Semantics of values
Type	USINT	Type of reset	See Table 91

The parameter Type for the Reset common service has the enumeration specifications shown in Table 91.

Table 91 – 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 92.

Table 92 – DeviceNet object class specific services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x4B	N/A	Optional	Allocate_Controller/Device_Connection_Set	Requests the use of the Predefined Controller/Device Connection Set
0x4C	N/A	Conditional[a]	Release_Controller/Device_Connection_Set	Indicates that the specified connections within the Predefined Controller/Device Connection Set are no longer desired. These connections shall be released (deleted)
0x4D	N/A	Conditional[b]	Clear_Diagnostics	Clears the diagnostic counters that are reported in attribute 12

[a] The Release_Controller/Device_Connection_Set service is required if the Allocate_Controller/Device_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_Controller/Device_Connection_Set, Release_Controller/Device_Connection_Set

These services are used to allocate and deallocate the Predefined Controller/Device Connection Set and are further specified in IEC 62026-3:2014, 5.5.

NOTE The Predefined Controller/Device Connection Set, Allocate_Controller/Device_Connection_Set service and Release_Controller/Device_Connection_Set service are called respectively Predefined Master/Slave Connection Set, Allocate_Master/Slave_Connection_Set service and Release Master/Slave_Connection_Set service in IEC 62026-3:2014.

A device that behaves as the client across the Predefined Controller/Device Connection Set is referred to as a Controller. A device that behaves as the server across the Predefined Controller/Device Connection Set is referred to as a Device. Within the bounds of the service descriptions to follow, a Controller and/or Device is viewed as a functional unit within a communicating device.

A device that wants to function as another's Controller shall first allocate the Predefined Controller/Device Connection Set within the Device. Only one Controller can have the Predefined Controller/Device Connection Set allocated at any given time. The entire Connection Set is allocated and the Controller uses a select subset of the connections from the set (i.e. Bit Strobe only, or Poll only). When a Controller wants to "give–up" its Device it releases all connections, causing the Device to "deallocate" the Predefined Controller/Device 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 88.

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 Port object shall be supported if the Connection Configuration object is supported and the device supports more than one Type 2 Port or the device supports a single Type 2 Port and the Port Number is not 2. The Port object is required because a client of the Connection Configuration object needs to be able to learn the Port Numbers of all the ports that can route the class 0/1 Forward Open services.

7.8.2 Revision history

The revision history of the Connection Configuration object is described in Table 93.

Table 93 – 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 94.

Table 94 – Connection Configuration object class attributes

Attri-bute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	NV	Revision	UINT	Revision of this object	Third revision, value = 3 (see 7.8.2)
2	Required	Get	NV	Max instance	UDINT	Maximum instance number	Value determined by node specifics
3	Required	Get	NV	Num instance	UDINT	Number of connections currently instantiated	
4	This class attribute is optional and is described in IEC 61158-5-2.						
5	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
6 **7**	These class attributes are optional and are described in IEC 61158-5-2.						
8	Required	Get	NV	Format number	UDINT	See 7.8.3.2	

Attri-bute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
9	Required	Get/Set	NV	Edit signature	UDINT	See 7.8.3.3	
10 – 32	Reserved						
33	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
34	Optional	Get	V	Scanner capabilities[c]	WORD	Bit 0, set to Scanner Mode (attribute 33) supported [d] Bit 1, set to Scanner Mode (attribute 33) 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 33 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 33 can be different than the value last set to attribute 33 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 9. This format value shall be specified in the format_number field of each instance attribute 9. Meaning of the format values is specified in Table 95.

Table 95 – 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. This value is initially 0.

Unless specified otherwise in the network communication profile, this value 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_Attributes_All data stream for each connection instance (lowest to highest) plus class attribute 1, class attribute 2, class attribute 3 and class attribute 8.

7.8.4 Instance attributes

7.8.4.1 General

The Connection Configuration object shall support the instance attributes as specified in Table 96.

Table 96 – Connection Configuration object instance attributes

Attri-bute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	V	Connection status	STRUCT of 4 octets	Connection status. Only two octets of extended status are included for error codes	See 7.8.4.2. NOTE For access to complete connection status, it is recommended that attribute 23 (Full Connection Status) be implemented as well.
				Gen_status	USINT	General status	The General Status value returned in the Forward_Open/ Large_Forward_Open response
				Reserved	USINT	Reserved	Shall be zero
				Ext_status	UINT	Extended status	If no extended status is returned, the Ext_status shall be zero
2	Required	Get/Set	NV	Connection flags	WORD	Connection flags	See 7.8.4.3
3	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	

Attri-bute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
				Major_rev	USINT	Major revision	
				Minor_rev	USINT	Minor revision	
4	Conditional	Get/Set	NV	CS data index number	UDINT		See 7.8.4.5
5	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	
6	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	
7	Required	Get	NV	Proxy Config 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	Proxy Config data	
8	Required	Get/Set	NV	Connection Name	STRING2	User-assigned connection name encoded in UNICODE	See 7.8.4.9
9	Required	Get/Set	NV	I/O mapping	STRUCT of variable size		See 7.8.4.10

Attri-bute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
				format_number	UINT	This number determines the format of this attribute	The format value shall match the value of class attribute 8
				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	
10	Required	Get/Set	NV	Target Config 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	Target Config data	
11	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	Minor revision	
12	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. The default value of this parameter is 0
13	Conditional[b]			Safety Parameters	See IEC 61784-3-2		
14	Conditional[b]			Safety Connection Parameter CRC			
15	Conditional[b]			Safety Configuration Instance			
16	Conditional[b]			Safety ID Allocation			
17	Conditional[b]			Safety Target Connection Serial Number			

Attri-bute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
18	Optional	Get/Set		Net Connection Parameters Attribute Selection	USINT	Selects between attribute 5 and attribute 19	See 7.8.4.13
19	Conditional[c]	Get/Set		Large Net Connection Parameters	STRUCT of 18 octets		See 7.8.4.14
				Conn_timeout	USINT	Connection Timeout Multiplier	
				Xport_class_and_trigger	SWORD	Transport Class and Trigger	
				Rpi_OT	UDINT	Originator to Target Requested Packet Interval	
				Net_OT	UDINT	Originator to Target large network connection parameters	
				Rpi_TO	UDINT	Target to Originator Requested Packet Interval	
				Net_TO	UDINT	Target to Originator large network connection parameters	
20	Conditional[b]			Format Type	See IEC 61784-3-2		
21	Conditional[b]			Format Status			
22	Conditional[b]			Max Fault Number			
23	Optional	Get	V	Full Connection Status	STRUCT of variable size	This attribute is the full connection status (handles extended status longer than a single WORD)	
				Gen_status	USINT	General status	General Status value returned in the Forward_Open/ Large_Forward_Open response
				Size of Ext_status	USINT	Number of WORD in Ext_status array	
				Ext_status	ARRAY of WORD	Extended status[d]	Empty, or Extended Status value returned in the Forward_Open/ Large_Forward_Open response

Attri-bute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
24	Optional	Get/Set	NV	Request Priority and Timeout	STRUCT of	See IEC 61158-6-2, 4.1.6.6. Default timeout value is vendor specific, 5 s (0x05, 156) recommended.	
				Priority/Time_tick [e]	SWORD		
				Time-out_ticks [e]	USINT		

[a] Attribute 12 shall be supported if conditional services Open_Connection (0x4C) and Close_Connection (0x4D) are supported. Attribute 12 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.

[c] Conditional attribute 19 is required if attribute 18 is supported, otherwise conditional attribute 19 is not allowed.

[d] If size_of_Ext_status is non-zero, then the first WORD of Ext_status shall be the same value as the Ext_status element in the Connection Status structure of attribute 1.

[e] This attribute is not included in the Get_Attributes_All response or Set_Attributes_All request.

7.8.4.2 Connection status

The values for the originator connection status are defined in Table 97.

Table 97 – 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 98.

Table 98 – 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 99.

Table 99 – Connection flags

Bit	Meaning
0	Connection 0 = Originator 1 = Target
1 – 3	O->T Real time transfer format 000 – 32-bit header format includes run/idle information 001 – Zero length data format indicates idle 010 – Modeless format – no run/idle notification 011 – Heartbeat format – no run/idle notification 100 – Reserved 101 – Safety format (see IEC 61784-3-2) 100 thru 111 – reserved for future use
4 – 6	T->O Real time transfer format 000 – 32-bit header format includes run/idle information 001 – Zero length data format indicates idle 010 – Modeless format – no run/idle notification 011 – Heartbeat format – no run/idle notification 100 – Reserved 101 – Safety format (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

Unless specified otherwise in the network communication profile, 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).

The xport_class_and_trig field shall be ignored for target instances.The device containing this CCO object shall determine if the similar Transport Type/Trigger parameter of the Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.14.

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).

The net_OT field shall be ignored for target instances, with the exception of the connection size. The device containing this CCO object shall determine if the similar O->T Network Connection Parameters parameter of the Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.1.

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).

The net_TO field shall be ignored for target instances, with the exception of the connection size. The device containing this CCO object shall determine if the similar T->O Network Connection Parameters parameter of the Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.1.

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/Large_Forward_Open requests). For target instances the open_path_size is used for matching the Connection_Path_Size received in a Forward_Open/Large_Forward_Open request.

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/Large_Forward_Open requests). For target instances the open_connection_path is used for matching the Connection_Path parameter received in a Forward_Open/Large_Forward_Open request.

7.8.4.8 Proxy Config data

This does not apply to target instances. The data specified in attributes 7 and 10 are concatenated (in that order) into a single data segment, then appended to the connection path in attribute 6 to form the complete path sent in the Forward_Open service of the Connection Manager object.

The configuration data may be split between attributes 7 and 10. When a connection is a non-proxied connection the convention that shall be followed is that all configuration data is placed in attribute 10 (Target Config). 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 7 (Proxy Config) and any configuration data intended for the proxied device is placed in attribute 10 (Target Config). Proxy Configdata 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.

The Connection Name shall not exceed 255 characters in length. See IEC 61158-5-2, 5.3.3.4 and IEC 61158-6-2 for details on the encoding for the STRING2 data type.

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 100 describes the structure of the mapping_data field of the I/O mapping attribute for each of the format numbers.

Table 100 – 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 Target Config data

This note does not apply to target instances. The data specified in attributes 7 and 10 are concatenated (in that order) into a single data segment, then appended to the connection path in attribute 6 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 10 or the configuration data may be split between attributes 7 and 10. 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 7 and any configuration data intended for the proxied device is placed in attribute 10. Target Config 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.

7.8.4.13 Net Connection Parameters Attribute Selection

The Net Connection Parameters Attribute Selection attribute selects between the use of instance attribute 5 (Net Connection Parameters) and attribute 19 (Large Net Connection Parameters).

Valid values are:

0 = indicates instance attribute 5 shall be used (default);

1 = indicates instance attribute 19 shall be used;

2-255 = reserved.

If this optional attribute is not supported attribute 5 shall be used.

7.8.4.14 Large Net Connection Parameters

7.8.4.14.1 conn_timeout

The conn_timeout is the value used for the connection timeout multiplier field defined in IEC 61158-6-2 (Large_Forward_Open request). conn_timeout is ignored for target instances.

7.8.4.14.2 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 (Large_Forward_Open request).

The xport_class and_trig field shall be ignored for target instances. The device containing this CCO object shall determine if the similar Transport Type/Trigger parameter of the Large_Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.14.

7.8.4.14.3 rpi_OT

The rpi_OT is the value used for the O->T RPI field defined in IEC 61158-6-2 (Large_Forward_Open request). rpi_OT is ignored for target instances.

7.8.4.14.4 net_OT

The net_OT is the value used for the O->T Network Connection Parameters field in IEC 61158-6-2 (Large_Forward_Open request).

The net_OT field shall be ignored for target instances, with the exception of the connection size. The device containing this CCO object shall determine if the similar O->T Network Connection Parameters parameter of the Large_Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.1.

7.8.4.14.5 rpi_TO

The rpi_TO is the value used for the T->O RPI field defined in IEC 61158-6-2 (Large_Forward_Open request). rpi_TO is ignored for target instances.

7.8.4.14.6 net_TO

The net_TO is the value used for the T->O Network Connection Parameters field in IEC 61158-6-2 (Large_Forward_Open request).

The net_TO field shall be ignored for target instances, with the exception of the connection size. The device containing this CCO object shall determine if the T->O Network Connection Parameters parameter of the Large_Forward_Open request is supported as specified in IEC 61158-6-2, 4.1.6.1.

7.8.5 Connection Configuration object change control

Changes to Connection Configuration object attributes can only be made after a Change_Start service has been successfully issued. Changes to Connection Configuration object attributes are applied after a Change_Complete service has been successfully issued.

The services which are valid during a change operation are listed in Table 101. A change operation is started by issuing a Change_Start service and completed by issuing a Change_Complete service.

Table 101 – Services valid during a change operation

Service code	Service name
0x02	Set_Attributes_All
0x08	Create
0x09	Delete
0x10	Set_Attribute_Single
0x15	Restore
0x4B	Kick_Timer
0x51	Change_Complete
0x52	Audit_Changes

7.8.6 Common services

7.8.6.1 General

The Connection Configuration object shall support the common services as specified in Table 102.

Table 102 – Connection Configuration object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Required	Required	Get_Attributes_All	Gets all attributes of the specified instance
0x02	N/A	Required	Set_Attributes_All	Sets all attributes of the specified instance
0x08	Required	N/A	Create	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

7.8.6.2 Get_Attributes_All

The structure of the Get_Attributes_All response at the class level is shown in Table 103.

Table 103 – Get_Attributes_All Response – class level

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	UINT	Revision [a]	1
2	UDINT	Max instance	
3	UDINT	Num instance	
8	UINT	Format number	
9	UDINT	Edit signature	

[a] The default value only applies for Revision 1 of this object. Newer revisions require that this attribute be implemented, which will override this default value.

The structure of the Get_Attributes_All response at the instance level is shown in Table 104.

Table 104 – Get_Attributes_All response – instance level

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	STRUCT of 4 octets	Connection status	
	USINT	Gen_status	
	USINT	Reserved	
	UINT	Ext_status	
2	WORD	Connection flags	
3	STRUCT of 8 octets	Target device ID	
	UINT	Vendor_id	
	UINT	Product_type	
	UINT	Product_code	
	USINT	Major_rev	
	USINT	Minor_rev	
4	UDINT	CS data index number	0xFFFFFFFF
5	STRUCT of 14 octets	Net connection parameters	
	USINT	Conn_timeout	
	SWORD	Xport_class_and_trig	
	UDINT	Rpi_OT	
	UINT	Net_OT	
	UDINT	Rpi_TO	
	UINT	Net_TO	
6	STRUCT of variable size	Connection path	
	USINT	Open_path_size	
	USINT	Reserved [a]	
	Padded EPATH	Open_connection_ path	
7	STRUCT of variable size	Proxy Config data	
	UINT	Config_data_size	
	ARRAY of octets	Config_data	
—	octet	Pad [b]	
10	STRUCT of variable size	Target Config data	

Attribute ID	Data type	Attribute name	Default value (if not implemented)
	UINT	Config_data_size	
	ARRAY of octets	Config_data	
—	octet	Pad [c]	
8	STRING2	Connection Name	
9	STRUCT of variable size	I/O mapping	
	UINT	format_number	
	UINT	mapping_data_ size	
	ARRAY of octets	mapping_data	
—	octet	Pad [d]	
11	STRUCT of 8 octets	Proxy device ID	
	UINT	Vendor_id	
	UINT	Product_type	
	UINT	Product_code	
	USINT	Major_rev	
	USINT	Minor_rev	
13-17	STRUCT of 55 octets See IEC 61784-3-2	Safety Parameters	(null) [e]
12	BOOL [f]	Connection disable	0
18	USINT	Net Connection Parameters Attribute Selection	0
19	STRUCT of 18 octets	Large Net Connection Parameters	
	USINT	Conn_timeout	0
	SWORD	Xport_class_and_trigger	0
	UDINT	Rpi_OT	0
	UDINT	Net_OT	0
	UDINT	Rpi_TO	0
	UDINT	Net_TO	0
20-22	STRUCT of variable size See IEC 61784-3-2	Additional Safety Parameters	(null) [e]

[a] This pad octet shall be zero.

[b] This pad octet shall only be included if the length of the Proxy Config Data (Attribute #7) is an odd number of octets in length.

[c] This pad octet shall only be included if the length of the Target Config Data (Attribute #10) is an odd number of octets in length.

[d] This pad octet shall only be included if the length of the I/O Mapping (Attribute #9) is an odd number of octets in length.

[e] The Safety Parameters and Additional Safety Parameters (variable length) blocks shall only be included in the request if either the O->T or T->O real time transfer format in the Connection Flags (Attribute #2) indicates the format is Safety.

[f] The BOOL type shall be encoded in bit 0 of a single octet. Bits 1-7 shall be zero

7.8.6.3 Set_Attributes_All

This service shall set all attributes associated with an existing instance and shall support the error codes shown in Table 105, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

Table 105 – Set_Attributes_All error codes

General status	Extended status	Error name	Description
0x0C	None	Object state conflict	The Connection Configuration object cannot accept the Set_Attribute_Single service when an edit session is not active

The structure of the Set_Attributes_All request is shown in Table 106.

Table 106 – Set_Attributes_All request

Attribute ID	Data type	Attribute name	Default value (if not implemented)
2	WORD	Connection flags	
3	STRUCT of 8 octets	Target device ID	
	UINT	Vendor_id	
	UINT	Product_type	
	UINT	Product_code	
	USINT	Major_rev	
	USINT	Minor_rev	
4	UDINT	CS data index number	0xFFFFFFFF
5	STRUCT of 14 octets	Net connection parameters	
	USINT	Conn_timeout	
	SWORD	Xport_class_and_trig	
	UDINT	Rpi_OT	
	UINT	Net_OT	
	UDINT	Rpi_TO	
	UINT	Net_TO	
6	STRUCT of variable size	Connection path	
	USINT	Open_path_size	
	USINT	Reserved [a]	
	Padded EPATH	Open_connection_ path	
7	STRUCT of variable size	Proxy Config data	
	UINT	Config_data_size	
	ARRAY of octets	Config_data	
—	octet	Pad [b]	
10	STRUCT of variable size	Target Config data	
	UINT	Config_data_size	
	ARRAY of octets	Config_data	
—	octet	Pad [c]	
8	STRING2	Connection Name	
9	STRUCT of variable size	I/O mapping	
	UINT	format_number	
	UINT	mapping_data_ size	
	ARRAY of octets	mapping_data	

Attribute ID	Data type	Attribute name	Default value (if not implemented)
—	octet	Pad [d]	
11	STRUCT of 8 octets	Proxy device ID	
	UINT	Vendor_id	
	UINT	Product_type	
	UINT	Product_code	
	USINT	Major_rev	
	USINT	Minor_rev	
13, 15, 16	STRUCT of 49 octets See IEC 61784-3-2	Safety Parameters	(null) [e]
12	BOOL [f]	Connection disable	0
18	USINT	Net Connection Parameters Attribute Selection	0
19	STRUCT of 18 octets	Large Net Connection Parameters	
	USINT	Conn_timeout	0
	SWORD	Xport_class_and_trigger	0
	UDINT	Rpi_OT	0
	UDINT	Net_OT	0
	UDINT	Rpi_TO	0
	UDINT	Net_TO	0
20, 22	STRUCT of variable size See IEC 61784-3-2	Additional Safety Parameters	(null) [e]

[a] This pad octet shall be zero.

[b] This pad octet shall only be included if the length of the Proxy Config Data (Attribute #7) is an odd number of octets in length.

[c] This pad octet shall only be included if the length of the Target Config Data (Attribute #10) is an odd number of octets in length.

[d] This pad octet shall only be included if the length of the I/O Mapping (Attribute #9) is an odd number of octets in length.

[e] The Safety Parameters and Additional Safety Parameters (variable length) blocks shall only be included in the request if either the O->T or T->O real time transfer format in the Connection Flags (Attribute #2) indicates the format is Safety.

[f] The BOOL type shall be encoded in bit 0 of a single octet. Bits 1-7 shall be zero

Data for unsupported attributes shall be accepted if the default values are sent. The service shall be rejected if non-default values are sent for unsupported attributes.

7.8.6.4 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 107 defines the request parameters for this service.

Table 107 – 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

Data for unsupported attributes shall be accepted if the default values are sent. The service shall be rejected if non-default values are sent for unsupported attributes.

This service shall read all attributes associated with an existing instance and shall support error codes shown in Table 108, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

Table 108 – Create error codes

General status	Extended status	Error name	Description
0x02	0x0001	Resource unavailable	The maximum number of instances already exist
	0x0002	Resource unavailable	Not enough memory on device
0x03	None	Invalid parameter value	The attribute count is invalid
0x08	None	Service not supported	Unimplemented service
0x0C	None	Object state conflict	The Connection Configuraton object cannot execute the Set_Attribute_Single service when an edit session is not active
0x0E	None	Attribute not settable	Attempt to set a read-only attribute
0x13	None	Not enough data	The request was too short or truncated
0x1C	None	Missing attribute list entry data	The required attribute was missing

7.8.6.5 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 109, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

Table 109 – Delete error codes

General status	Extended status	Error name	Description
0x0C	None	Object state conflict	The Connection Configuraton object cannot execute the Delete service when an edit session is not active

7.8.6.6 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 110, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

Table 110 – Restore error codes

General status	Extended status	Error name	Description
0x0C	None	Object state conflict	The Connection Configuraton object cannot execute the Restore service when an edit session is not active

7.8.7 Class specific services

7.8.7.1 General

The Connection Configuration object shall support the class specific services as specified in Table 111.

Table 111 – 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[a]	Conditional[a]	Open_Connection	Opens connections
0x4D	Conditional[a]	Conditional[a]	Close_Connection	Closes connections
0x4E	Conditional[a]	Conditional[a]	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

[a] 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.

7.8.7.2 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.

7.8.7.3 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.

7.8.7.4 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.

7.8.7.5 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.

7.8.7.6 Change_Start

This service shall

a) signal the beginning of an edit session;

b) 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 112, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

Table 112 – Change_Start error codes

General status	Extended status	Error name	Description
0x0C	None	Object state conflict	The Connection Configuraton object cannot execute the Restore service when an edit session is not active
0x10	None	Device state conflict	The device is in a state that prevents starting an edit session

7.8.7.7 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 113.

Table 113 – Get_Status service parameter

Parameter	Data type	Description
Starting Instance	UDINT	Starting instance number

The response parameters are defined for this service in Table 114.

Table 114 – 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	Connection Configuration instance number
	USINT	General status
	USINT	Reserved, shall be 0
	UINT	Extended status

This service shall support the error codes shown in Table 115, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

Table 115 – Get_Status service error codes

General status	Extended status	Error name	Description
0x03	None	Invalid Parameter Value	The attribute count is invalid

7.8.7.8 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 116.

Table 116 – 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 117, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

Table 117 – Change_Complete service error codes

General status	Extended status	Error name	Description
0x02	None	Resource unavailable	Indicates that there is not enough memory on the host device to support the specified configuration.
0x0C	None	Object state conflict	An edit session is not active
0x10	None	Device state conflict	The device is in a state that prevents completing an edition session

7.8.7.9 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 118.

Table 118 – 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 119, in addition to error codes listed in IEC 61158-6-2, 4.1.11.2.1.

Table 119 – Audit_Changes service error codes

General status	Extended status	Error name	Description
0x02	None	Resource unavailable	Indicates that there is not enough memory on the host device to support the specified configuration.
0x0C	None	Object state conflict	An edit session is not active
0x10	None	Device state conflict	The device is in a state that prevents completing an audit

7.8.8 Behavior

Figure 25 summarizes the process of creating, editing, and deleting connections.

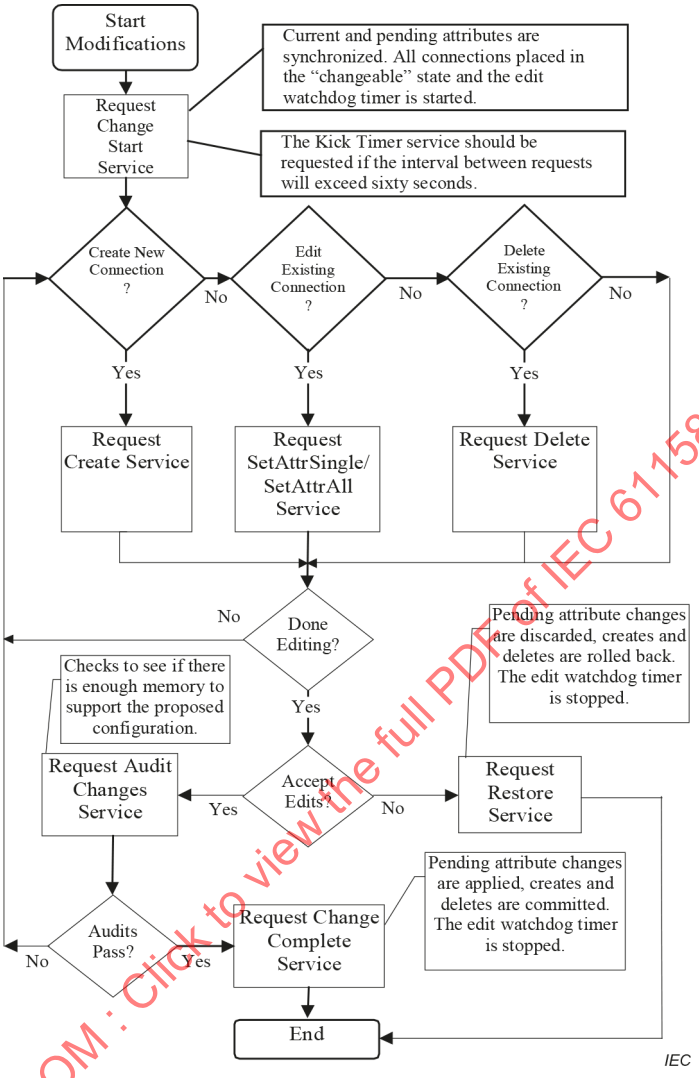


Figure 25 – Connection Configuration object edit flowchart

7.9 DLR object

7.9.1 Overview

The Device Level Ring (DLR) object provides the Type 2 application-level configuration and status information interface for the DLR protocol. The DLR protocol is a layer 2 protocol that enables the use of an Ethernet ring topology. The DLR protocol is fully specified in Clause 10.

The DLR object shall be implemented in all multi-port Type 2 Ethernet devices that support the DLR protocol.

Devices shall implement one instance of the DLR object for each pair of DLR ring ports supported.

NOTE Support for the DLR protocol on multiple pairs of ports is future enhancement that is at the present time undefined.

7.9.2 Revision history

Table 120 shows the revision history for the DLR object.

Table 120 – Revision history

Revision	Description	Status/Edition
1	Initial revision of this object definition	Obsolete
2	Instance attribute 10 changed to required, instance attribute 12 added, and corresponding changes in Get_Attributes_All response. Added Restart_Sign_On object specific service.	Obsolete
3	Added conditional attribute 13 for Redundant Gateway Configuration. Added conditional attribute 14 for Redundant Gateway Status. Added conditional attribute 15 for Active Gateway Address. Added conditional attribute 16 for Active Gateway Precedence. Added redundant gateway capability bit in Capabilities Flags attribute (12). Added Flush_Tables frame support capability bit in Capability Flags attribute (12). All Revision 3 and higher devices are required to support the Flush_Tables and Learning_Update frames. Added Get_Attributes_All responses for redundant gateway devices.	Required (November 2012)
4	Added conditional Ring Port 1 and 2 Ethernet Link Object Instance attributes (17 & 18). Added conditional Enable attribute (19).	Conditional[a] (November 2016)
[a] Required for devices that support attributes 17, 18 or 19.		

7.9.3 Class attributes

The DLR Object shall support the class attributes as specified in Table 121.

Table 121 – DLR object class attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Conditional[a]	Get	NV	Revision	UINT	Revision of this object	Third revision, value = 3
2 to 7	These class attributes are optional and are described in IEC 61158-5-2.						
[a] Required if the Revision value is greater than 1.							

7.9.4 Instance attributes

7.9.4.1 General

The DLR object shall support the instance attributes as specified in Table 122.

Table 122 – DLR object instance attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	V	Network Topology	USINT	Current network topology mode	0 – Linear 1 – Ring See 7.9.4.2
2	Required	Get	V	Network Status	USINT	Current status of network	See 7.9.4.3
3	Conditional [a]	Get	V	Ring Supervisor Status	USINT	Ring supervisor active status flag	See 7.9.4.4
4	Conditional [a]	Set	NV	Ring Supervisor Config	STRUCT of	Ring Supervisor configuration parameters	See 7.9.4.5
				Ring Supervisor Enable	BOOL	Ring supervisor enable flag	TRUE – the device is configured as a ring supervisor. FALSE – the device is configured as a normal ring node Default=FALSE
				Ring Supervisor Precedence	USINT	Precedence of a ring supervisor in network with multiple ring supervisors	Numerically higher value indicates higher precedence Default=0
				Beacon Interval	UDINT	Duration of ring beacon interval	Beacon interval in microseconds. Default=400 µs
				Beacon Timeout	UDINT	Duration of ring beacon timeout	Beacon timeout in microseconds. Default=1 960 µs
				DLR VLAN ID	UINT	VLAN ID to use in ring protocol messages	Value range is 0 to 4 094 Default=0
5	Conditional [a]	Set	V	Ring Faults Count	UINT	Number of ring faults since power up	See 7.9.4.6

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
6	Conditional [a]	Get	V	Last Active Node on Port 1	STRUCT of	Last active node at the end of chain through port 1 of active ring supervisor during ring fault	See 7.9.4.7
					UDINT	Device IP Address	A value of 0 indicates no IP Address has been configured for the device. Initial value shall be 0
					USINT[6]	Device MAC Address	Ethernet MAC address
7	Conditional [a]	Get	V	Last Active Node on Port 2	STRUCT of	Last active node at the end of chain through port 2 of active ring supervisor during ring fault	See 7.9.4.8
					UDINT	Device IP Address	A value of 0 indicates no IP Address has been configured for the device
					USINT[6]	Device MAC Address	Ethernet MAC address
8	Conditional [a]	Get	V	Ring Protocol Participants Count	UINT	Number of devices in ring protocol participants list	See 7.9.4.9
9	Conditional [a]	Get	V	Ring Protocol Participants List	ARRAY of	List of devices participating in ring protocol	See 7.9.4.10
					STRUCT of		
					UDINT	Device IP Address	A value of 0 indicates no IP Address has been configured for the device
					USINT[6]	Device MAC Address	Ethernet MAC address
10	Required	Get	V	Active Supervisor Address	STRUCT of	IP and/or MAC address of the active ring supervisor	See 7.9.4.11
					UDINT	Supervisor IP Address	A value of 0 indicates no IP Address has been configured for the device
					USINT[6]	Supervisor MAC Address	Ethernet MAC address
11	Conditional [a]	Get	V	Active Supervisor Precedence	USINT	Precedence value of the active ring supervisor	See 7.9.4.12
12	Required	Get	NV	Capability Flags	DWORD	Describes the DLR capabilities of the device	See 7.9.4.13

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
13	Conditional [b]	Set	NV	Redundant Gateway Config	STRUCT of	Redundant Gateway configuration parameters	See 7.9.4.14
				Redundant Gateway Enable	BOOL	Redundant gateway enable flag	TRUE indicates the device is configured as a redundant gateway. FALSE indicates device is not configured as a redundant gateway. Default=FALSE
				Gateway Precedence	USINT	Precedence of a gateway in network with multiple gateways	Numerically higher value indicates higher Precedence. Default=0
				Advertise Interval	UDINT	Duration of active gateway Advertise Interval	Advertise Interval in microseconds. Default=2 000 µs
				Advertise Timeout	UDINT	Duration of active gateway Advertise Timeout	Advertise Timeout in microseconds. Default=5 000 µs
				Learning Update Enable	BOOL	Learning Update Enable flag	TRUE indicates all DLR nodes will send Learning_Update frame after gateway switchover. FALSE indicates DLR nodes will not send Learning_Update frame after gateway switchover. Default=TRUE
14	Conditional [b]	Get	V	Redundant Gateway Status	USINT	Current status of the gateway device	See 7.9.4.15
15	Conditional [b]	Get	V	Active Gateway Address	STRUCT of	IP and/or MAC address of the active gateway device	See 7.9.4.16
					UDINT	Active gateway IP Address	A value of 0 indicates no IP Address has been configured for this device
					USINT[6]	Active gateway MAC Address	Ethernet MAC address
16	Conditional [b]	Get	V	Active Gateway Precedence	USINT	Precedence value of the active gateway	See 7.9.4.17
17	Conditional [c]	Set [d]	NV	Ring Port 1 Ethernet Link Object Instance	UINT	Instance ID of associated Ethernet Link object instance for ring port 1	0 – Unconfigured 1 to 1 023 – Instance number of

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
18	Conditional [c]	Set [d]	NV	Ring Port 2 Ethernet Link Object Instance	UINT	Instance ID of associated Ethernet Link object instance for ring port 2	associated Ethernet Link object Instance 1 024 to 65 534 – Reserved 65 535 – Not supported Default is vendor specific. See 7.9.4.18
19	Conditional [d]	Set	NV	DLR Enable	BOOL	Indicates whether the DLR processing as configured for this instance is enabled or disabled	0 – All DLR operations disabled 1 – DLR operations enabled See 7.9.4.19

[a] Shall be implemented for devices capable of functioning as a ring supervisor. Shall not be implemented by non-supervisor devices.

[b] Shall be implemented for devices capable of functioning as a redundant gateway. Shall not be implemented by non-redundant gateway devices.

[c] Required if the device supports more than two Ethernet Link object instances that are associated with external Ethernet ports, otherwise optional.

[d] Set is optional. A device may provide statically defined pairs or provide configuration of ring pairs through vendor specific means.

[e] Required if the device instantiates more than one resiliency protocol object at the same time (e.g. DLR and PRP), or if attributes 17 and 18 are supported and can be changed by the user, else optional.

7.9.4.2 Network topology

The Network Topology attribute indicates the current network topology mode. A value of 0 indicates "Linear" topology. A value of 1 indicates "Ring" topology. The value of the attribute shall correspond to the DLR State.

When a supervisor-capable device is enabled as a ring supervisor, the Network Topology attribute shall always indicate "Ring". When a supervisor-capable device is not enabled as a ring supervisor, or is not a supervisor-capable device, the device shall initially indicate "Linear", then shall transition between "Ring" and "Linear" modes.

7.9.4.3 Network status

The Network Status attribute provides current status of the network based the device's view of the network, as specified in the DLR behavior in 10.6. Table 123 specifies the possible values for the Network Status.

Table 123 – Network Status values

Network Status value	Description
0	Normal operation in both Ring and Linear Network Topology modes
1	Ring Fault. A ring fault has been detected. Valid only when Network Topology is Ring
2	Unexpected Loop Detected. A loop has been detected in the network. Valid only when the Network Topology is Linear [a]
3	Partial Network Fault. A network fault has been detected in one direction only. Valid only when Network Topology is Ring and the node is the active ring supervisor
4	Rapid Fault/Restore Cycle. A series of rapid ring fault/restore cycles has been detected, according to the criteria given in 10.6.3.5. Similar to the Partial Network Fault status, the supervisor remains in a state with forwarding blocked on its ring ports. The condition shall be cleared explicitly via the "Clear Rapid Faults" service

[a] If the device is capable of detecting reception of frames that it sent, it shall report the "Unexpected Loop Detected" value.

7.9.4.4 Ring supervisor status

The Ring Supervisor Status attribute indicates the device's status as a ring supervisor. Table 124 specifies the possible values for the Ring Supervisor Status.

Table 124 – Ring Supervisor Status values

Ring Supervisor Status value	Description
0	The device is functioning as a backup supervisor
1	The device is functioning as the active ring supervisor
2	The device is functioning as a normal ring node (supervisor not enabled)
3	The device is functioning in a non-DLR topology (supervisor not enabled, and no other supervisor is present)
4	The device cannot support the currently operating ring parameters (Beacon Interval and/or Beacon Timeout)

7.9.4.5 Ring supervisor config

7.9.4.5.1 General

The Ring Supervisor Config attribute contains the configuration parameters needed for ring operation: Supervisor Precedence, Beacon Interval, Beacon Timeout, VLAN ID, Supervisor Enable/Disable.

If the device is the active supervisor, changes to the attribute shall be applied immediately. When the Supervisor Precedence, Beacon Interval, Beacon Timeout or VLAN ID are changed on the active ring supervisor, the ring supervisor shall cease sending Beacon frames for two beacon timeout periods then shall resume sending Beacon frames using the new parameters.

Backup ring supervisors shall obtain values for the Beacon Interval, Beacon Timeout and VLAN ID from the Beacon frame sent by the active ring supervisor, and shall store those values in non-volatile storage. If the obtained values cannot be supported by the device (e.g., Beacon Interval too small), the device shall set the Ring Supervisor Status attribute as noted in the attribute description, shall report a minor recoverable fault via the Identity object Status attribute (5), and shall not take over as active supervisor after a ring reconfiguration.

When the Ring Supervisor Config attribute is modified on a backup supervisor, the behavior depends on the backup supervisor's new precedence value compared to the active supervisor's precedence value:

- new backup precedence value is greater than the current active ring supervisor's precedence or of equal precedence with numerically higher MAC address than active supervisor MAC address – backup shall immediately begin sending Beacon frames with the new parameters;
- new backup precedence value is less than the active supervisor's precedence or of equal precedence with numerically lower MAC address than active supervisor MAC address – modification to the Beacon Interval, Beacon Timeout, and VLAN ID shall be ignored.

Attempts to set invalid Ring Supervisor Config values shall result in error code 0x09 (Invalid attribute value) returned from the Set service, regardless of whether the device is an active or backup supervisor.

7.9.4.5.2 Ring supervisor enable

The Ring Supervisor Enable item enables or disables the ring supervisor function in a supervisor-capable device. A value of TRUE enables the supervisor function. A value of FALSE disables the supervisor function. The default value is FALSE.

7.9.4.5.3 Ring supervisor precedence

The Ring Supervisor Precedence item contains the user-assigned precedence value given to the ring supervisor. When multiple ring supervisors are enabled, the precedence value allows the user to configure the order in which the configured supervisors select the active supervisor.

The precedence value for the ring supervisor shall be chosen from the range 0 to 255, with numerically higher values indicating higher precedence. The default value shall be 0.

When more than one supervisor is enabled, the supervisor with highest precedence becomes active ring supervisor, in accordance with 10.6.2. If multiple supervisors have the same precedence, the supervisor with the numerically higher MAC address becomes the active supervisor.

7.9.4.5.4 Beacon interval

The Beacon Interval item contains the interval in microseconds that the ring supervisor shall use in generating beacon frames. Per the DLR protocol specification in Clause 10, the default value shall be 400 µs. Supervisors shall support a range from 400 µs to 100 ms. Supervisors may support a Beacon Interval smaller than 400 µs, but this is not required. The absolute minimum Beacon Interval is 100 µs.

7.9.4.5.5 Beacon timeout

The Beacon Timeout item contains the number of microseconds the ring supervisor shall wait for a beacon frame before declaring a beacon timeout.

The default value shall be 1 960 µs, which is based on a nominal network size of 50 nodes and 100 Mbit/s, full-duplex operation (see the performance calculations in 10.12). The user may wish to change the Beacon Timeout for other exceptional network circumstances (e.g., very large networks or very small high-performance motion networks).

The Beacon Timeout shall be at least 2 times the Beacon Interval value. If the Beacon Interval is changed and the Beacon Timeout becomes less than 2 times the Beacon Interval, the supervisor shall adjust the Beacon Timeout to be 2 times the Beacon Interval.

Supervisors shall support a range from 800 µs to 500 ms. Supervisors may support a Beacon Timeout of smaller than 800 µs but this is not required. The absolute minimum Beacon Timeout is 200 µs.

7.9.4.5.6 DLR VLAN ID

The DLR VLAN ID contains the VLAN ID to be used in the DLR protocol frames. The DLR VLAN ID shall be used for all DLR protocol frames originated by the device, when the device is operating as the active ring supervisor. Devices that are not the active ring supervisor shall use the VLAN ID obtained from the active supervisor's frames (see Clause 10 for additional details).

The VLAN ID value shall be in the range 0 to 4 094. The default value shall be 0 (indicating no VLAN).

7.9.4.6 Ring Faults Count

The Ring Faults Count attribute contains the number of times since power up that the device has detected a ring fault, as either active or backup supervisor. If the Ring Supervisor Enable is set to FALSE, the Ring Faults Count shall be set to 0. The Ring Faults Count rolls over to 0 after it reaches its maximum value.

The attribute may also be reset to 0 via the Set_Attribute_Single service. Values other than 0 shall result in an error response.

7.9.4.7 Last active node on port 1

The Last Active Node on Port 1 attribute contains the IP address and/or Ethernet MAC address of the last node reachable through port 1 of an active ring supervisor. The value of the attribute is obtained via the Link_Status/Neighbor Status frames, as specified in 10.10.6.

On transition to FAULT_STATE, this attribute shall remain clear until the supervisor receives Link/Neighbor Status information.

On transition from FAULT_STATE to NORMAL_STATE, the value of the attribute shall be retained, to aid in diagnosing the previous ring fault.

The initial values of IP address and Ethernet MAC address shall be 0. When the device is not enabled as a ring supervisor, or is operating as the backup supervisor, the IP address and Ethernet MAC address shall be 0.

7.9.4.8 Last active node on port 2

The Last Active Node on Port 2 attribute contains the IP address and/or Ethernet MAC address of the last node reachable through port 2 of an active ring supervisor. The value of the attribute is obtained via the Link_Status/Neighbor Status frames, as specified in 10.10.6.

On transition to FAULT_STATE, this attribute shall remain clear until the supervisor receives Link/Neighbor Status information.

On transition from FAULT_STATE to NORMAL_STATE, the value of the attribute shall be retained, to aid in diagnosing the previous ring fault.

The initial values of IP address and Ethernet MAC address shall be 0. When the device is not enabled as a ring supervisor, or is operating as the backup supervisor the IP address and Ethernet MAC address shall be 0.

7.9.4.9 Ring protocol participants count

This attribute contains the number of members in the Ring Protocol Participants List attribute. The count and the list are gathered by the active ring supervisor through Sign_On frame as specified in 10.10.9.

If the device is not the active supervisor, the attribute shall be set to 0.

7.9.4.10 Ring protocol participants list

The Ring Protocol Participants List attribute contains the list of ring nodes participating in ring protocol. The participants list is gathered by the active ring supervisor via the Sign_On frame (see 10.10.9).

Since the size of the Ring Protocol Participants List could be large, depending on the number of nodes participating in the ring, this attribute shall be accessible with the Get_Member service.

Clients may elect to use the Get_Attribute_Single service to read the Ring Protocol Participants List. If the participants list is too large, the DLR object shall return error code 0x11 (Reply Data Too Large).

If the device is not active supervisor, status code 0x0C (Object State Conflict) shall be returned.

If the number of participants received as a result of the Sign_On process exceeds the capacity of the supervisor's Ring Protocol Participants List, the last entry in the list shall be all 0xFF's (0xFFFFFFFFFFFFFFFFFFFF).

7.9.4.11 Active supervisor address

This attribute contains the IP address and/or Ethernet MAC address of the active ring supervisor. The initial values of IP address and Ethernet MAC address shall be 0, until the active ring supervisor is determined.

7.9.4.12 Active supervisor precedence

This attribute contains the precedence value of the active ring supervisor. The initial value shall be 0, until the active ring supervisor is determined.

7.9.4.13 Capability flags

The Capability Flags describe the DLR capabilities of the device, as specified in Table 125.

Table 125 – Capability flags

Bit(s):	Name	Definition
0	Announce-based Ring Node[a]	Set if device's ring node implementation is based on processing of Announce frames (see 10.5)
1	Beacon-based Ring Node[a]	Set if device's ring node implementation is based on processing of Beacon frames (see 10.5)
2-4	Reserved	Shall be set to zero.
5	Supervisor Capable	Set if device is capable of providing the supervisor function.
6	Redundant Gateway Capable	Set if device is capable of providing the redundant gateway function.
7	Flush_Table frame Capable	Set if device is capable of supporting the Flush_Tables frame.
8-31	Reserved	Shall be set to zero.

[a] Bits 0 and 1 are mutually exclusive. Exactly only one of these bits shall be set in the attribute value that a device reports.

7.9.4.14 Redundant gateway config

7.9.4.14.1 General

The Redundant Gateway Config attribute contains the configuration parameters needed for redundant gateway operation: Gateway enable/disable, Gateway Precedence, Advertise Interval, Advertise Timeout and Learning Update enable/disable.

If the device is the active gateway, changes to the attribute shall be applied immediately. When the Gateway Precedence, Advertise Interval, Advertise Timeout or Learning Update enable/disable are changed on the active gateway, the active gateway shall cease sending Advertise frames for 1,5 times old Advertise Timeout period and then shall resume sending Advertise frames using the new parameters.

Backup gateway devices shall obtain values for the Advertise Interval, Advertise Timeout and Learning Update enable/disable from the Advertise frame sent by the active gateway, and shall store those values in non-volatile storage. If the obtained values cannot be supported by the device (e.g., Advertise Interval too small), the device shall set the Redundant Gateway Status attribute as noted in the attribute description, shall report a minor recoverable fault via the Identity object Status attribute (5), and shall not take over as active gateway after a gateway reconfiguration.

When the Redundant Gateway Config attribute is modified on a backup gateway, the behavior depends on the backup gateway's new Precedence value compared to the active gateway's Precedence value.

- New backup Precedence value is greater than the current active gateway's Precedence or of equal Precedence with numerically higher MAC address than active gateway MAC address: backup shall immediately begin sending Advertise frames with the new parameters (see 10.11.4).
- New backup Precedence value is less than the active gateway's Precedence or of equal Precedence with numerically lower MAC address than active gateway MAC address: Advertise Interval, Advertise Timeout and Learning Update enable/disable shall be ignored.

Attempts to set invalid Redundant Gateway Config values shall result in error code 0x09 (Invalid attribute value) returned from the set service, regardless of whether the device is an active or backup gateway.

7.9.4.14.2 Redundant gateway enable

The Redundant Gateway Enable item enables or disables the gateway function in a redundant gateway-capable device. A value of TRUE enables the gateway function. A value of FALSE disables the gateway function. The default value is FALSE.

7.9.4.14.3 Gateway precedence

The Gateway Precedence item contains the user-assigned Precedence value given to a gateway. When multiple gateways are enabled, the Precedence value allows the user to configure the order in which the configured gateways select the active gateway.

The gateway Precedence shall be chosen from the range 0 to 255, with numerically higher values indicating higher Precedence. The default value shall be 0.

When more than one gateway is enabled the gateway with highest Precedence becomes active gateway, in accordance with Clause 10. If multiple gateways have the same Precedence, the gateway with the numerically higher MAC address becomes the active gateway.

7.9.4.14.4 Advertise interval

The Advertise Interval item contains the interval in microseconds that the active gateway shall use in generating Advertise frames. Per the DLR protocol specification in Clause 10, the default value shall be 2 000 µs. Gateways shall support a range from 1 000 µs to 100 ms. Gateways may support an Advertise Interval smaller than 1 000 µs, but this is not required. The absolute minimum Advertise Interval is 200 µs.

7.9.4.14.5 Advertise timeout

The Advertise Timeout item contains the number of microseconds a backup gateway shall wait for an Advertise frame before declaring an Advertise Timeout.

The default value shall be 5 000 µs, which is based on a nominal network size of 50 nodes and 100 Mbit/s, full-duplex operation (refer to the Performance Calculations in Clause 10). The user may wish to change the Advertise Timeout for exceptional network circumstances (e.g., very large networks or very small high-performance motion networks).

The Advertise Timeout shall be at least 2,5 times the Advertise Interval value. If the Advertise Interval is changed and the Advertise Timeout becomes less than 2,5 times the Advertise Interval, the gateway shall adjust the Advertise Timeout to be 2,5 times the Advertise Interval.

Gateways shall support a range from 2 500 µs to 500 ms. Gateways may support an Advertise Timeout of smaller than 2 500 µs but this is not required. The absolute minimum Advertise Timeout is 500 µs.

7.9.4.14.6 Learning update enable

The Learning Update Enable item enables/disables transmission of Learning_Update frames by DLR nodes when they receive Flush_Tables frame from active gateway. This parameter is encoded by active gateway in Flush_Tables frame and sent to DLR nodes. The Learning_Update frames from DLR devices accelerate the new network topology learning by all non-DLR switches outside DLR network after an active gateway switchover. A value of TRUE enables the learning update function. A value of FALSE disables the learning update function. The default value is TRUE.

7.9.4.15 Redundant gateway status

The Redundant Gateway Status attribute indicates the device's status as a gateway. Table 126 shows the possible values.

Table 126 – Redundant Gateway Status values

Redundant Gateway Status value	Description
0	Indicates the device is functioning as a non-gateway DLR node (gateway not enabled)
1	Indicates the device is functioning as a backup gateway
2	Indicates the device is functioning as the active gateway
3	Indicates gateway fault state due to loss of communication on uplink port
4	Indicates the device cannot support the currently operating gateway parameters (Advertise Interval and/or Advertise Timeout)
5	Indicates gateway fault state due to partial network fault (see 10.9.4.8)
6-255	Reserved

7.9.4.16 Active gateway address

This attribute contains the IP address and/or Ethernet MAC address of the active gateway device. The initial values of IP address and Ethernet MAC address shall be 0, until the active gateway is determined.

7.9.4.17 Active gateway precedence

This attribute contains the Precedence value of the active gateway device. The initial value shall be 0, until the active gateway is determined.

7.9.4.18 Ring Port 1 and 2 Ethernet Link Object Instance

If the value of either of these attributes is 0, DLR cannot operate. Therefore, if the default value of either of these attributes is 0, the DLR Enable attribute (19) default value shall be disabled (0).

If these attributes are settable, then attempts to change these attributes when DLR is enabled (i.e. DLR Enable is enabled (1)) shall return an error with a General Status of Object State Conflict (0x0C).

If these attributes have a valid configuration and DLR is enabled (i.e. DLR Enable is enabled (1)), then a client shall first disable DLR before they can be changed. The changes only become active after DLR is enabled. See the Recommendation in 7.9.4.19 for recommended workflow when changing DLR ring port assignments.

The value of 65 535 (0xFFFF) is used only for the Get_Attributes_All response when these attributes are not supported.

7.9.4.19 DLR Enable

This attribute determines whether DLR operations are enabled for the two ports associated with this DLR instance as reflected in the Ring Port 1 and 2 Ethernet Link Object Instances attributes (17 and 18), if supported.

A value of 0 means that all DLR functions are disabled. A value of 1 means that the associated ports are connected to the same network segment and shall provide ring node functionality if beacons are detected, and may provide ring supervisor and/or redundant gateway functions as supported by the device.

When the default value for either Ring Port 1 or 2 Ethernet Link Object Instance attributes (17 and 18) is zero (i.e. the port associations are unconfigured), the default value for this attribute shall be disabled (0); otherwise the default value shall be enabled (1).

When this attribute is not supported, DLR behavior is enabled.

If either of the ring port attributes (17 and 18) have a value of zero (0) and an attempt is made to set this attribute to enabled (1), then an error with a General Status of Object State Conflict (0x0C) shall be returned. See 7.9.4.18.

Recommendation:

Since disabling DLR can create a storm, if this node is the Ring Supervisor and there is no active Backup Supervisor in the ring, then the user should be advised to first break the ring. If the user wishes to change the port assignments (attributes #17,18) the following work flow should be used:

- disable or unplug one of the currently configured ring ports;
- disable DLR;
- change one or both of the port associations;
- enable DLR;
- enable or plug in ring cable(s) in accordance with new port assignments.

7.9.5 Diagnostic connection points

Two diagnostic connection points are defined for the DLR object class at the instance level. One of these connection points is required, as appropriate based on device capabilities, if the Standard Network Diagnostic Assembly is implemented (see IEC 61158-6-2, 4.1.8.4.1.4).

NOTE See IEC 61158-5-2, 6.1.6 for further information about Diagnostic Connection Points.

The format of the DLR diagnostic connection points are as specified in Table 127 and Table 128.

Table 127 – DLR connection point 1, Standard Network Diagnostics

Attribute ID	Attribute name	Data type	Attribute size	Size of structure
2	Network Status	USINT	1 octet	8 octets
N/A	56 bits pad, shall be zeros		7 octets	

Table 128 – DLR connection point 2, Standard Network Diagnostics

Attribute ID	Attribute name	Data type	Attribute size	Size of structure
2	Network Status	USINT	1 octet	8 octets
3	Ring Supervisor Status	USINT	1 octet	
5	Ring Faults Count	UINT	2 octets	
N/A	32 bits pad, shall be zeros		4 octets	

7.9.6 Common services

7.9.6.1 General

The common services implemented by the DLR object are specified in Table 129.

Table 129 – DLR object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Required	Get_Attributes_All	Returns multiple attributes in numerical order
0x0E	Optional	Required	Get_Attribute_Single[c]	Returns the contents of the specified attribute or connection point
0x10	N/A	Conditional [a]	Set_Attribute_Single	Modifies a single attribute
0x18	N/A	Conditional [b]	Get_Member	Returns members of attribute
0x1D	N/A	Conditional [d]	Get_Connection_Point_Member_List	Used to get member paths of a Connection Point structure

[a] This service shall be implemented for devices capable of functioning as a ring supervisor and/or redundant gateway.

[b] Required for access to the Ring Protocol Participants List. The member services extended protocol for multiple sequential members shall be supported.

[c] This service shall support the use of a connection point in the service path when Diagnostic Connection Point(s) are implemented (see 7.9.5).

[d] This service is required if Diagnostic Connection Point(s) are implemented.

7.9.6.2 Get_Attributes_All response

7.9.6.2.1 Class level

At the class level, the Get_Attributes_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

7.9.6.2.2 Instance level

At the instance level the Get_Attributes_All response varies depending on the revision of the object and the device's capabilities. The response format can be differentiated by the response length.

The response length for an Object Revision 1, non-supervisor device is 2 octets. The response format is specified in Table 130.

Table 130 – Get_Attributes_All Response – Object Revision 1, non supervisor device

Attribute ID	Data type	Name
1	USINT	Network Topology
2	USINT	Network Status

The response length for an Object Revision 1, supervisor-capable device is 50 octets. The response format is specified in Table 131.

Table 131 – Get_Attributes_All Response – Object Revision 1, supervisor-capable device

Attribute ID	Data type	Name
1	USINT	Network Topology
2	USINT	Network Status
3	USINT	Ring Supervisor Status
4	STRUCT of	Ring Supervisor Config
	BOOL	Ring Supervisor Enable
	USINT	Ring Supervisor Precedence
	UDINT	Beacon Interval
	UDINT	Beacon Timeout
	UINT	DLR VLAN ID
5	UINT	Ring Faults Count
6	STRUCT of	Last Active Node on Port 1
	UDINT	Device IP Address
	USINT[6]	Device MAC Address
7	STRUCT of	Last Active Node on Port 2
	UDINT	Device IP Address
	USINT[6]	Device MAC Address
8	UINT	Ring Protocol Participants Count
10	STRUCT of	Active Supervisor Address
	UDINT	Device IP Address
	USINT[6]	Device MAC Address
11	USINT	Active Supervisor Precedence

The response length for an Object Revision 2, non-supervisor–capable device is 16 octets. The response format is specified in Table 132.

Table 132 – Get_Attributes_All Response – Object Revision 2, non supervisor device

Attribute ID	Data type	Name
1	USINT	Network Topology
2	USINT	Network Status
10	STRUCT of	Active Supervisor Address
	UDINT	Device IP Address
	USINT[6]	Device MAC Address
12	DWORD	Capability Flags

In all other cases, the Get_Attributes_All response (see Table 133) shall contain the instance attributes 1 through 8 and 10 through 19, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default value specified.

Responses lengths are as follows:

- The response length for an Object Revision 2, supervisor–capable device is 54 octets.

- The response length for an Object Revision 3, non-supervisor, non-gateway device is 54 octets.
- The response length for an Object Revision 3, supervisor, non-gateway device is 54 octets.
- The response length for an Object Revision 3, gateway-capable device is 77 octets.
- The response length for an Object Revision 4 device not supporting attributes 17 through 19 is either 54 (non-gateway) or 77 (gateway-capable) octets.
- The response length for an Object Revision 4 device supporting attributes 17 through 19 is 82 octets.

Table 133 – Get_Attributes_All Response – All other cases

Attribute ID	Data type	Name	Default value (if not implemented)
1	USINT	Network Topology	
2	USINT	Network Status	
3	USINT	Ring Supervisor Status	0xFF – Not applicable
4	STRUCT of	Ring Supervisor Config	
	BOOL	Ring Supervisor Enable	0
	USINT	Ring Supervisor Precedence	0
	UDINT	Beacon Interval	0
	UDINT	Beacon Timeout	0
	UINT	DLR VLAN ID	0
5	UINT	Ring Faults Count	0
6	STRUCT of	Last Active Node on Port 1	
	UDINT	Device IP Address	0
	USINT[6]	Device MAC Address	All zero
7	STRUCT of	Last Active Node on Port 2	
	UDINT	Device IP Address	0
	USINT[6]	Device MAC Address	All zeroes
8	UINT	Ring Protocol Participants Count	0xFFFF – Not applicable
10	STRUCT of	Active Supervisor Address	
	UDINT	Device IP Address	
	USINT[6]	Device MAC Address	
11	USINT	Active Supervisor Precedence	0
12	DWORD	Capability Flags	
13	STRUCT of	Redundant Gateway Configuration	
	BOOL	Redundant Gateway Enable	0
	USINT	Gateway Precedence	0
	UDINT	Advertise Interval	0
	UDINT	Advertise Timeout	0
	UINT	Learning Update Enable	0
14	USINT	Redundant Gateway Status	0xFF
15	STRUCT of	Active Gateway Address	
	UDINT	Device IP Address	0
	USINT[6]	Device MAC Address	All zeros
16	USINT	Active Gateway Precedence	0

Attribute ID	Data type	Name	Default value (if not implemented)
17	UINT	Ring Port 1 Ethernet Link Object Instance	0xFFFF
18	UINT	Ring Port 2 Ethernet Link Object Instance	0xFFFF
19	BOOL	DLR Enable	1

7.9.7 Class specific services

7.9.7.1 General

The DLR object shall support the class specific services specified in Table 134.

Table 134 – DLR object class specific services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x4B	N/A	Conditional [a]	Verify_Fault_Location	Causes ring supervisor to verify fault location by issuing Locate_Fault ring protocol message to ring nodes and update Last Active Node 1 and Last Active Node 2 attributes
0x4C	N/A	Conditional [a]	Clear_Rapid_Faults	Clears the Rapid Fault/Restore Cycle Detected condition in the ring supervisor, allowing the supervisor to return to normal operation
0x4D	N/A	Conditional [a]	Restart_Sign_On	Restart Sign On process and refresh DLR participants list
0x04E	N/A	Conditional [b]	Clear_Gateway_Partial_Fault	Clears the partial network fault condition in the gateway device, allowing the gateway to return to normal operation

[a] This service shall be implemented for devices capable of functioning as a ring supervisor.

[b] This service shall be implemented for devices capable of functioning as a redundant gateway.

7.9.7.2 Verify_Fault_Location

The Verify_Fault_Location service shall cause an active ring supervisor to verify a ring fault location by retransmitting the Locate_Fault frame to ring nodes (see 10.10.7). The Last Active Node 1 and Last Active Node 2 attributes shall be updated based on the response to the Locate_Fault frame.

There are no parameters for either the Verify_Fault_Location request or response.

If the Verify _Fault_Location service is received when the supervisor is not enabled, or is currently the backup supervisor, or is the active supervisor but not in fault state, status code 0x0C (Object State Conflict) shall be returned, and the Last Active Node 1 and Last Active Node 2 attributes shall be set to 0.

7.9.7.3 Clear_Rapid_Faults

The Clear_Rapid_Faults service shall clear the condition where the ring supervisor has detected a cycle of rapid ring fault/restore (as defined in 10.6.3.5). Upon clearing the condition, the ring supervisor shall return to normal operation.

If the Clear_Rapid_Faults service is received when the supervisor is not enabled, or is currently the backup supervisor, or is the active supervisor but not in the rapid fault/restore condition, status code 0x0C (Object State Conflict) shall be returned.

7.9.7.4 Restart_Sign_On

The Restart_Sign_On service shall restart Sign On process (see 10.6.2.3 Sign On for more information) by active ring supervisor, if it is not currently in progress.

If the Restart_Sign_On service is received when the supervisor is not enabled, or is currently the backup supervisor, or is the active supervisor but not in the NORMAL_STATE, status code 0x0C (Object State Conflict) shall be returned. If the Restart_Sign_On service is received when a prior Sign On process is in progress, success shall be returned and the request shall be ignored.

7.9.7.5 Clear_Gateway_Partial_Fault

The Clear_Gateway_Partial_Fault service shall clear the condition where the gateway has detected a partial network fault (as defined in Clause 10). Upon clearing the condition, the gateway shall execute state machine as specified in Clause 10.

If the Clear_Gateway_Partial_Fault service is received when the gateway is not enabled, or is not in partial network fault condition, status code 0x0C (Object State Conflict) shall be returned.

7.10 QoS object

7.10.1 Overview

Quality of Service (QoS) is a general term that is applied to mechanisms used to treat traffic streams with different relative priorities or other delivery characteristics. Standard QoS mechanisms include IEEE Std 802.1Q-2018 (Ethernet frame priority) and Differentiated Services (DiffServ) in the TCP/IP protocol suite.

The QoS object provides a means to configure certain QoS-related behaviors in Type 2 Ethernet devices.

The QoS Object is required for devices that support sending Type 2 Ethernet messages with non-zero DiffServ code points (DSCP), or sending Type 2 Ethernet messages in IEEE Std 802.1Q-2018 tagged frames.

7.10.2 Revision History

The revision history of the QoS object is described in Table 135.

Table 135 – QoS object revision history

Revision	History
01	Initial Definition

7.10.3 Class attributes

The QoS object shall support the class attributes as specified in Table 136.

Table 136 – QoS object class attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	Revision	UINT	Revision of this object	Current value for this attribute = 1
2 to 7	These class attributes are optional and are described in IEC 61158-5-2.					

7.10.4 Instance Attributes

7.10.4.1 General

The QoS object shall support the instance attributes as specified in Table 137.

Table 137 – QoS object instance attributes

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Conditional[a]	Set	NV	802.1Q Tag Enable	USINT	Enables or disables sending IEEE Std 802.1Q-2018 frames on Type 2 Ethernet and IEC 61588 messages	A value of 0 indicates tagged frames disabled. A value of 1 indicates tagged frames enabled. The default value shall be 0
2	Conditional[b]	Set	NV	DSCP PTP Event	USINT	DSCP value for PTP (IEC 61588) event messages	See 7.10.4.3
3	Conditional[b]	Set	NV	DSCP PTP General	USINT	DSCP value for PTP (IEC 61588) general messages	See 7.10.4.3
4	Required	Set	NV	DSCP Urgent	USINT	DSCP value for Type 2 transport class 0/1 Urgent priority messages	See 7.10.4.3
5	Required	Set	NV	DSCP Scheduled	USINT	DSCP value for Type 2 transport class 0/1 Scheduled priority messages	See 7.10.4.3
6	Required	Set	NV	DSCP High	USINT	DSCP value for Type 2 transport class 0/1 High priority messages	See 7.10.4.3
7	Required	Set	NV	DSCP Low	USINT	DSCP value for Type 2 transport class 0/1 low priority messages	See 7.10.4.3
8	Required	Set	NV	DSCP Explicit	USINT	DSCP value for Type 2 explicit messages (transport class 2/3 and UCMM) and all other Type 2 Ethernet encapsulation messages	See 7.10.4.3

[a] Required if the device supports sending IEEE Std 802.1Q-2018 frames.

[b] Required if the device supports the Time Sync object.

7.10.4.2 802.1Q Tag Enable

The 802.1Q Tag Enable attribute enables or disables sending IEEE Std 802.1Q-2018 frames on Type 2 Ethernet and IEC 61588 messages. When the attribute is enabled, the device shall send IEEE Std 802.1Q-2018 frames for all Type 2 Ethernet and IEC 61588 messages.

A value of 1 indicates enabled. A value of 0 indicates disabled. The default value for the attribute shall be 0. A change to the value of the attribute shall take effect the next time the device restarts.

Devices shall always use the corresponding DSCP values regardless of whether IEEE Std 802.1Q-2018 frames are enabled or disabled.

7.10.4.3 DSCP value attributes

Attributes 2 to 8 contain the DSCP values that shall be used for the different types of Type 2 Ethernet traffic.

The format of the DSCP value within the IP header is specified in IEC 61158-6-2, 11.6.4. Since the DSCP field has a size of 6 bits, the valid range of values for these attributes is 0 to 63. The DSCP value, if placed directly in the ToS field in the IP header, shall be shifted left 2 bits.

Table 138 specifies the default DSCP values and traffic usages.

Table 138 – Default DCSP values and usages

Attribute	Traffic type usage	Default DSCP
DSCP PTP Event	PTP (IEC 61588) event messages	59 ('111011')
DSCP PTP General	PTP (IEC 61588) general messages	47 ('101111')
DSCP Urgent	Transport class 0/1 messages with Urgent priority	55 ('110111')
DSCP Scheduled	Transport class 0/1 messages with Scheduled priority	47 ('101111')
DSCP High	Transport class 0/1 messages with High priority	43 ('101011')
DSCP Low	Transport class 0/1 messages with Low priority	31 ('011111')
DSCP Explicit	UCMM messages, transport class 2/3 messages, and all other Type 2 Ethernet encapsulation messages	27 ('011011')

A change to the value of the above attributes shall take effect the next time the device restarts.

7.10.5 Common services

7.10.5.1 General

The QoS object shall support the common services as specified in Table 139.

Table 139 – QoS object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	N/A	Get_Attributes_All	Returns multiple attributes in numerical order
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] Required if any class attributes are implemented.

7.10.5.2 Get_Attributes_All response (class level)

The Get_Attributes_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

7.11 Port object

7.11.1 Overview

The Port object describes the communication interfaces that are present on the device and visible to Type 2 networks. A tool/originator can use the Port object to learn the types of subnets and/or backplanes the device contains, which ones support Type 2 routing, as well as which object provides access to its logical link-specific functionality. In addition a tool/originator can use the Port object to learn the Type 2 routing capabilities (see Port object instance attribute 10).

The Port object refers to a logical link/network interface on the device, i.e. the interface used to address it. The logical link address is defined by the corresponding network communication profile.

In many cases, a single Port instance is used to directly reflect a physical port. In some cases, multiple Port instances share a single physical port (e.g. Port instances for two different Ethernet-based fieldbuses may share the same Ethernet interface). And there are cases where there are multiple physical ports for a single Port object instance (e.g. a dual-port Ethernet-based Type 2 device with DLR).

The Port Object provides the port name, port number (used for Type 2 routing) within the device, and other port information for each communication interface on the device.

Implementation requirements are as follows.

1) One instance shall exist for each Type 2 routable port.
2) One instance shall exist for each non-routable Type 2 port that has the same Logical Link Object Class ID (instance attribute 3) as another port.
3) The Port object is required if another object definition requires it.
4) Instances may exist for non-routable ports.
5) Instances may exist for devices that support a single port. A single port device may support routing by allowing an incoming packet to be routed back out the port, in this case one instance shall exist for the port.

7.11.2 Revision History

The revision history of the Port object is described in Table 140.

Table 140 – Port object revision history

Revision	History
01	Initial definition
02	Instance Attribute 10 added – Port routing capabilities

7.11.3 Class attributes

The Port Object shall support the class attributes as specified in Table 141.

Table 141 – Port object class attributes

Attribute ID	Need in implementation	Access rule	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is two (2).
2	Required	Get	Max Instance	UINT	Maximum instance number	
3	Required	Get	Num Instances	UINT	Number of ports currently instantiated	
4 to 7	These class attributes are optional and are described in IEC 61158-5-2.					
8	Required	Get	Entry Port	UINT	Returns the instance of the Port object that describes the port through which this request entered the device	
9	Required	Get	Port Instance Info	ARRAY of STRUCT of	Array of structures containing instance attributes 1 and 2 from each instance	The array is indexed by instance number starting with zero, up to the maximum instance number. The values for instance 0 and any non-instantiated instances shall be zero.
			Port Type	UINT	Enumerates the type of port	See instance attribute #1
			Port Number	UINT	Type 2 port number associated with this port	See instance attribute #2

7.11.4 Instance attributes

7.11.4.1 General

The Port object shall support the instance attributes as specified in Table 142.

Table 142 – Port object instance attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	NV	Port Type	UINT	Enumerates the type of port	See 7.11.4.2
2	Required	Get	NV	Port Number	UINT	Type 2 port number associated with this port	See 7.11.4.2
3	Required	Get	NV	Logical Link Object	STRUCT of		See 7.11.4.2
				Path Length	UINT	Number of 16 bit words in the following path [a]	
				Link Path	Padded EPATH	Logical path segments that identify the object for this port. For example, this could be the TCP/IP Interface object.	
4	Required	Get	NV	Port Name	SHORT_STRING	String which names the communication interface. The maximum number of characters in the string is 64. For example, this can be "Port A".	See 7.11.4.4
5	Optional	Get	NV	Port Type Name	SHORT_STRING	String which names the port type. The maximum number of characters in the string is 64.	See 7.11.4.2
6	Optional	Set	NV	Port Description	SHORT_STRING	String which describes the port. The maximum number of characters in the string is 64. For example, this can be "Product Line 22".	See 7.11.4.5
7	Required	Get	NV	Port Number and Node Address	Padded EPATH	This is a single port segment containing the port number of this port and the link address of this device on this port	See 7.11.4.6
8	Conditional [b]	Get	NV	Port Node Range	STRUCT of		See 7.11.4.7
				Minimum Node Number	UINT	Minimum node number on port	
				Maximum Node Number	UINT	Maximum node number on port	

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
9	Conditional [c]	Get	NV	Chassis Identity	Packed EPATH	Electronic key of network/chassis this port is attached to. This attribute shall be a single Logical Electronic Key segment.with Format 4 of the Logical Electronic Key segment. The Vendor ID, Device Type, Product Code, Major Revision and Minor Revision fields shall not be 0. The Compatibility field shall be 0 (indicating match).	See 7.11.4.8
10	Required	Get	NV	Port Routing Capabilities	DWORD	Bit string that defines the routing capabilities of this port	See 7.11.4.9
11	Conditional [d]	Get	NV	Associated Communication Objects	STRUCT of	List of communication object instances associated with this Port Object instance	See 7.11.4.10
					USINT	Number of entries in array	
					Array of STRUCT of		
					USINT	Number of 16 bit words in the following path	
					Padded EPATH	Logical path segments that identify an associated communication object instance	

[a] The Path Length may be 0 if there is no logical link object associated with this port instance. If a logical link object exists for the port instance, the path shall be specified.

[b] If a device can report its port characteristics within the range allowed (e.g. Type 2 MAC ID) then it shall support this attribute. Otherwise (e.g. Internet IP address) it shall not support this attribute.

[c] If this port is attached to a chassis then this attribute is optional, otherwise it is not allowed.

[d] If there is more than one Port Object instance that points to the same Logical Link object class ID (see instance attribute 3), this attribute is required, otherwise this attribute is optional for this instance.

7.11.4.2 Port Type, Logical Link Object and Port Type Name

The vendor assigns values to these three attributes to indicate the type of the port, whether or not it supports routing, and whether it provides a link specific object to make link specific functionality visible to Type 2.

When a port provides routing, the corresponding Port Type (1 to 65 534), Logical Link Object and Port Type Name values assigned to the supported technology in Table 143 shall be used.

When a port does not provide routing, the Port Type value shall be set to 0 and the corresponding Logical Link Object and Port Type Name values assigned to the supported technology in Table 143 shall be used.

When a port cannot report its type information based on its state (e.g. the user shall choose which protocol it should support), the Port Type value shall be set to 65 535. See Port Type value 65 535 in Table 143 for the corresponding Logical Link Object and Port Type Name values.

The Port Type values and their associated Logical Link Object classes and Port Type Name values are listed in Table 143.

Table 143 – Port Type and associated Logical Link Object classes and Port Type Name values

Port Technology [c]	Port Type (attribute 1)	Logical Link Object (attribute 3)		Port Type Name (attribute 5)
		Logical Link Object Name	Class Code	
Any technology below (1 to 65 534), but routing is not supported	0	Logical Link object assigned to underlying technology or null.	Class code of assigned logical link object (if implemented)	Name assigned to underlying technology (1 to 65 534) below.
Reserved for legacy use [a]	1	Vendor Specific or null	Vendor Specific (if implemented)	Vendor assigned
ControlNet™ (Type 2)[5]	2	ControlNet object	0xF0	ControlNet
ControlNet™ with media redundancy (Type 2)	3	ControlNet object	0xF0	ControlNet Redundant
EtherNet/IP™ (Type 2)[5]	4	TCP/IP Interface object	0xF5	EtherNet/IP [b]
DeviceNet™ (Type 2)[5]	5	DeviceNet object	0x03	DeviceNet
Reserved for legacy use [a]	6 to 99	Vendor Specific or null	Vendor Specific (if implemented)	Vendor Assigned
Vendor Specific Types 100 to 199	100 to 199	Vendor Specific or null	Vendor Specific (if used)	Vendor Assigned
CompoNet™[5]	200	CompoNet Link object	0xF7	CompoNet
Modbus/TCP (Type 15)[6]	201	TCP/IP Interface object	0xF5	Modbus/TCP
Modbus/SL (Type 15)[6]	202	Modbus Serial Link object	0x46	Modbus/SL
SERCOS III (Type 19)[7]	203	Sercos III Link object	0x4C	SERCOS III
HART (Type 9)[8]	204	null	None	HART

5 ControlNet™, EtherNet/IP™, DeviceNet™ and CompoNet™ are trade names of ODVA, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance with this profile does not require use of the trade names ControlNet™, EtherNet/IP™, DeviceNet™ or CompoNet™. Use of the trade names ControlNet™, EtherNet/IP™, DeviceNet™ or CompoNet™requires permission from ODVA, Inc.

6 Modbus is a trademark of Schneider Automation Inc registered in the United States of America and other countries. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance with this profile does not require use of the trademark Modbus. Use of the trademark Modbus requires permission from Schneider Automation Inc.

7 SERCOS is a trade name of sercos international e.V. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance with this profile does not require use of the registered trademark. Use of the trade name requires permission of the trade name holder.

8 HART is a registered trademark of FieldComm Group. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance with this profile does not require use of the registered trademark. Use of the trademark HART requires permission of the trademark holder.

Port Technology [c]	Port Type (attribute 1)	Logical Link Object (attribute 3)		Port Type Name (attribute 5)
		Logical Link Object Name	Class Code	
IO-Link (IEC 61131-9)[9]	205	null	None	IO-Link
Reserved for future use	206 to 65 534	Reserved for future use		
Any technology above (1 to 65 534), but the type of port is user configurable	65 535	Logical Link object assigned to underlying technology or null.	Class code of assigned logical link object (if implemented)	Unconfigured port

[a] Port types marked as "Reserved for legacy use" indicate port types that were defined prior to the publication of this document. They are not defined in this document. Devices should not implement these port types without prior knowledge of the legacy usage.

[b] This port type was formerly known as TCP/IP.

[c] Port technologies with port types 200 to 205 correspond to technologies with Type 2 interfaces as specified in dedicated volumes of the ODVA specifications.

The logical link object path value (attribute 3) shall consist of one logical class segment and one logical instance segment. The maximum size is 12 octets. See definition of logical segment in IEC 61158-6-2, 4.1.9.4.

7.11.4.3 Port Number

Manufacturer assigns a unique value to identify each communication port. Value 0 is reserved and cannot be used.

Value 1 is reserved for a backplane port. Devices with no backplane port or a backplane that does not support Type 2 communications shall not have a Port Number 1. The Port Number for the backplane port of a device that supports a single backplane port shall be 1. The Port Number for one of the backplane ports of a device that supports multiple backplane ports shall be 1 and the others can be any other valid port number.

The Port Number value is encoded in the Port segment defined in IEC 61158-6-2, 4.1.9.3, and is used for routing.

7.11.4.4 Port Name

This attribute is the vendor assigned name of the communications interface associated with this instance.

This value is unique for each communications interface. If multiple Type 2 ports use the same communications interface, they shall each have the same Port Name value. Likewise, a Type 2 port shall not have the same Port Name value as any other Type 2 port if they do not share the same communications interface.

When there is a one-to-one relationship between a communications interface and a physical network port and the port has a user visible label on the product, the value should be equivalent to the label (e.g. use the textual name for a symbol).

When there is a one-to-many relationship between an internal communication interface and multiple physical network ports and each port has a user visible label on the product, this

9 IO-Link is a trade name of the IO-Link Consortium. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the trade name holder or any of its products. Compliance with this document does not require use of the registered trade name. Use of the trade name IO-Link requires permission of the trade name holder.

attribute should report the internal interface name (typically, the logical link object instance associated with each physical port reports the user visible label names in this case).

7.11.4.5 Port Description

The default value of the Port Description attribute shall be an empty string. This attribute is used by a client to add a description to their port.

7.11.4.6 Port Number and Node Address

The Port Number and Node Address value shall be a Port Segment containing the Port Number of this port and the Link Address of this device on the port. The encoded port number shall match the value presented in attribute 2.

For devices that do not have a Link Address on this port (for example, a Type 2 Ethernet port with no assigned IP address) or for Port Type 0 (no Type 2 routing), the Link Address portion of the Port Segment shall be zero length.

EXAMPLE For Port Number 2, a zero-length Link Address would be indicated by 0x12 0x00.

See IEC 61158-6-2, 4.1.9.3 for details on the Port Segment including examples.

7.11.4.7 Port Node Range

This attribute indicates the lowest and highest link address on the Port Type (attribute 1) that this port is connected to. This attribute can be used by a browsing tool to limit the range of link addresses browsed.

EXAMPLE For a 17 slot backplane, this attribute would have the values 0 and 16, for a network supporting MAC IDs between 0 and 63 this attribute would have the values 0 and 63.

7.11.4.8 Chassis Identity

This attribute can be used to indicate the identity of the chassis that this port is currently plugged into. This attribute is useful when browsing. For example, the browser can identify the chassis that the device is plugged into, can load the chassis EDS file and associated icon, and use this identification to determine what additional modules can be plugged into this chassis. If this attribute is not implemented, then browsing clients will not have a standardized way of identifying the chassis, and will have to represent the chassis generically or use vendor-specific methods.

This attribute should be used to indicate the Identity of the chassis as accurately as possible. However, due to technical limitations of some chassis implementations, it could be impossible or not practical to accurately determine certain attributes of the chassis, including version, slot ranges, and specific part numbers. In this case, the vendor may choose to include the identity of a generic chassis or perhaps the largest available chassis.

7.11.4.9 Port Routing Capabilities

This attribute indicates the specific routing capabilities of this port.

The Port Routing Capabilities attribute shall be all 0's if no routing is supported.

See Table 144 for the definition of the routing capability bits. If the bit is set, the port supports the specified routing function. In this table, "incoming" refers to routing requests received on this port to any port on the device, and "outgoing" refers to routing requests received from any port out this port.

Table 144 – Port Routing Capabilities attribute bit definitions

Bit	Routing Capability
0	Routing of incoming Unconnected Messaging supported
1	Routing of outgoing Unconnected Messaging supported
2	Routing of incoming Transport Class 0/1 connections supported
3	Routing of outgoing Transport Class 0/1 connections supported
4	Routing of incoming Transport Class 2/3 connections supported
5	Routing of outgoing Transport Class 2/3 connections supported
6	Valid only for some network communication profiles
7 – 31	Reserved

7.11.4.10 Associated Communication Objects, Attribute 11

This attribute shall include any open and/or vendor specific objects that impact the operation of the communications path associated with this Port object instance. Each entry in the Array of Padded EPATH shall specify a single class segment followed by a single instance segment. If this attribute is supported, every Communication object associated with this port instance shall be included. See IEC 61158-6-2, 4.1.10.2.1 for the list of Associated Communication objects.

EXAMPLE

The following list is an example of open objects that are classified as Associated Communication objects:

- ControlNet object
- Keeper object
- Scheduling object
- TCP/IP Interface object
- Ethernet Link object
- DeviceNet object
- DLR object
- QoS object
- PRP/HSR Protocol object
- PRP/HSR Nodes Table object
- SERCOS III Link object

If there are multiple Port object instances that have the same Logical Link Object Class ID (attribute 3), none of the objects in the Array of Padded EPATH are allowed to be instance 1, i.e. instance 1 shall not be implemented for those object classes. This prevents old clients that could have assumed instance 1 for an Associated Communication object from retrieving incorrect information.

The following Associated Communications Objects are excluded from the requirement to use instance numbers other than 1, because tools that existed prior to the definition of the Associated Communications Objects attribute make no assumption about instance 1 for these objects:

- QoS Object – 0x48 (if there is only one instance that applies to all TCP ports)
- Ethernet Link Object – 0xF6

See Figure 26 and Table 145 for an example of a device with two Type 2 Ethernet ports, both supporting DLR, to see how the Instance 1 numbering restrictions are applied.

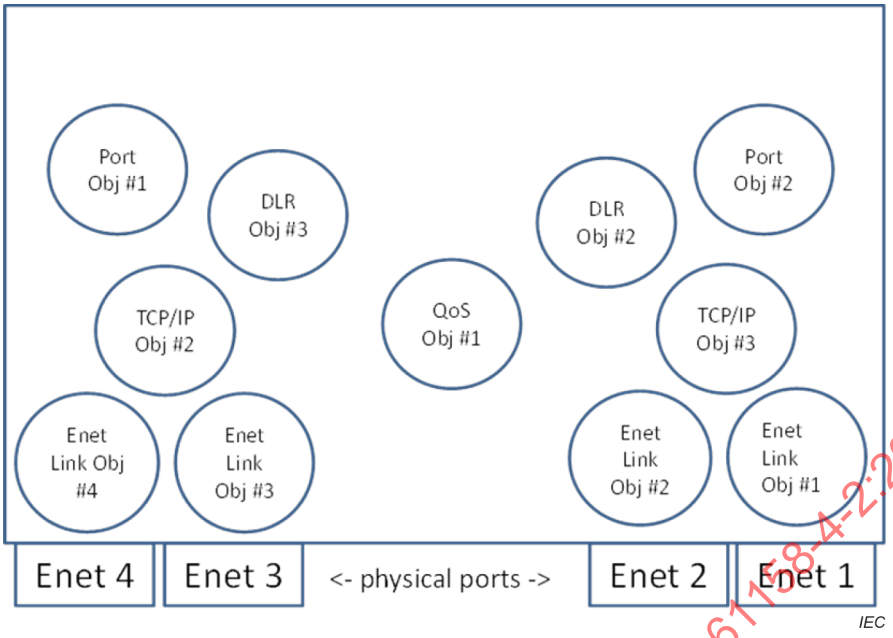


Figure 26 – Communication objects diagram for example device

Table 145 – Contents of Associated Communication objects attribute 11 for the two Port object instances of the example device

Port Object Instance 1 Attribute 11	Port Object Instance 2 Attribute 11	Comments
TCP/IP object (0xF5) Instance 2	TCP/IP object (0xF5) Instance 3	Both Type 2 Ports use the same underlying technology, i.e. the TCP/IP Object, so Instance restrictions apply (since old tools will mislead user if instance 1 was present). This is true even if one port is Type 2 Ethernet and other port is Modbus/TCP.
DLR object (0x47) Instance 3	DLR object (0x47) Instance 2	Cannot use instance 1
Ethernet Link object (0xF6) Instance 3 Instance 4	Ethernet Link object (0xF6) Instance 1 Instance 2	Instance 1 allowed since old tools do not assume port of entry is instance 1.
QoS object (0x48) Instance 1	QoS object (0x48) Instance 1	Instance 1 allowed since single instance applies to both ports. If QoS object only pertained to one of the Type 2 Ethernet ports, or if there was one QoS object for each port, then Instance 1 could not have been used.

7.11.5 Common services

7.11.5.1 General

The common services implemented by the Port object are specified in Table 146.

Table 146 – Port object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Optional	Get_Attributes_All	Returns the contents of all attributes of the class/instance
0x0E	Required	Required	Get_Attribute_Single	Returns the contents of the specified class/instance attribute
0x10	N/A	Conditional [a]	Set_Attribute_Single	Modifies an attribute

[a] This service shall be implemented if any of the settable class/instance attribute is supported.

7.11.5.2 Get_Attributes_All response

7.11.5.2.1 Class level

The structure of the Get_Attributes_All response at the class level is specified in Table 147.

Table 147 – Get_Attributes_All response– class level

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	UINT	Revision	1 [a]
2	UINT	Max Instance	
3	UINT	Number of Instances	
8	UINT	Entry Port	
9	ARRAY of	Port Instance Info	
	STRUCT of		
	UINT	Port Type	
	UINT	Port Number	

[a] The default value only applies for Revision 1 of this object. Newer revisions require that this attribute be implemented, which will override this default value.

7.11.5.2.2 Instance level

The structure of the Get_Attributes_All response at the instance level is specified in Table 148.

Table 148 – Get_Attributes_All response– instance level

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	UINT	Port Type	
2	UINT	Port Number	
	STRUCT of	Logical Link Object	
3	UINT	Path Length	
	Padded EPATH	Link Path	
4	SHORT_STRING	Port Name	
7	Padded EPATH	Port Number and Node Address	

7.12 PRP/HSR Protocol object

7.12.1 Overview

The PRP/HSR Protocol object provides a configuration and diagnostic interface. The Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR) protocol are OSI layer 2 Link Redundancy Entity (LRE) protocols that provide high availability through a dual attachment.

An LRE is an entity at layer 2 that hides port redundancy from the upper layers, by forwarding to the upper layers the frames received from the active redundant ports as if they came from a single port, and by forwarding to the active redundant ports a frame coming from the upper layers.

PRP and HSR are fully specified in IEC 62439-3, and also briefly described in Clause 11. The PRP/HSR Protocol object provides the Type 2 application-level interface to these protocols.

7.12.2 Revision history

The revision history of the PRP/HSR Protocol object is described in Table 149.

Table 149 – Revision history

Revision	Reason for object definition update
1	Initial revision of this object definition, with PRP only (obsolete)
2	Revision of this object definition integrating HSR with PRP (obsolete)
3	Add additional diagnostics (Instance Attributes 17-20) and counter reset functionality

7.12.3 Class attributes

The PRP/HSR Protocol object shall support the class attributes as specified in Table 150.

Table 150 – Class attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	NV	Revision	UINT	Revision of object	The current value assigned to this attribute is 3
2 to 7	These class attributes are optional and are described in IEC 61158-5-2.						

7.12.4 Instance attributes

7.12.4.1 General

The PRP/HSR Protocol object shall support the instance attributes as specified in Table 151. These instance attributes correlate to the MIB object definitions described in IEC 62439-3:2016, Clause 7.

Table 151 – Instance attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	NV	LRE Enable	BOOL	LRE Enable Flag	TRUE indicates that LRE is Enabled FALSE indicates that LRE is Disabled Default=FALSE
2	Required	Get	NV	Node Type	UINT	Operating mode of Node	Default = 1 See 7.12.4.2
3	Required	Get	NV	Node Name	SHORT_STRING	Human readable representation of Name of the LRE	Vendor defined ASCII characters. A length of 0 shall indicate no Node Name is configured. The string content should be the same as the MIB Object ID, if any is provided.
4	Required	Get	NV	Version Name	SHORT_STRING	Human readable representation of version Name of the LRE	Vendor defined ASCII characters. Maximum length is 32 characters. The string content should be the same as the MIB Object ID, if any is provided. A length of 0 is not allowed for this attribute.
5	Required	Get is required Set is optional	NV	LRE MAC Address	ETH_MAC_ADDR	Specifies the MAC address to be used by LRE	
6	Required	Get is required Set is optional	NV	Duplicate Discard	UINT	Specifies whether the duplicate discard algorithm is used at reception. The RCT shall always be inserted by a LRE	doNotDiscard (0) discard (1) Default = 1
7	Required	Get/Set	NV	Transparent Reception	UINT	If 0, the RCT is removed from the frame by the LRE before forwarding to the upper layers	removeRCT (0), passRCT (1); Default = 0;

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
8	Required	Get is required Set is optional	V	LRE Interface Counters	STRUCT of		See 7.12.4.3
				Transmit Count A	UDINT	Number of frames sent over Port A that are HSR tagged or RCT appended	
				Transmit Count B	UDINT	Number of frames sent over Port B that are HSR tagged or RCT appended	
				Transmit Count C	UDINT	Number of frames sent over the interlink of the Redundancy Box or to the DANP application interface	
				Receive Count A	UDINT	Number of frames received on Port A that are HSR tagged or RCT appended	
				Receive Count B	UDINT	Number of frames received on Port B that are HSR tagged or RCT appended	
				Receive Count C	UDINT	Number of frames received on the interlink of the Redundancy Box or the DANP application interface	
				Wrong LAN A Count	UDINT	Number of frames with the wrong LAN identifier received on LRE Port A.	Only applicable to PRP ports
				Wrong LAN B Count	UDINT	Number of frames with the wrong LAN identifier received on LRE Port B.	Only applicable to PRP ports
				Wrong LAN C Count	UDINT	Number of frames with the wrong LAN identifier received on LRE Port C.	Only applicable to HSR RedBoxes in HSR-PRP

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
9	Required	Get is required Set is optional	V	LRE Duplicate Detection Counters	STRUCT of		See 7.12.4.4
				Entries Unique Count A	UDINT	Number of Entries in the duplicate detection mechanism on port A for which no duplicate was received.	
				Entries Unique Count B	UDINT	Number of Entries in the duplicate detection mechanism on port B for which no duplicate was received.	
				Entries Duplicate Count A	UDINT	Number of Entries in the duplicate detection mechanism on port A for which one single duplicate was received.	
				Entries Duplicate Count B	UDINT	Number of Entries in the duplicate detection mechanism on port B for which one single duplicate was received.	
				Entries Multiple Count A	UDINT	Number of Entries in the duplicate detection mechanism on port A for which more than one duplicate was received.	
				Entries Multiple Count B	UDINT	Number of Entries in the duplicate detection mechanism on port B for which more than one duplicate was received.	
10	Conditional [a]	Get	V	Proxy Nodes Count	UDINT	Number of nodes in the Proxy Nodes Table	
11	Conditional [a]	Get, Get_ Member(s)	V	Proxy Nodes Table	ARRAY of		
				Proxy Node MAC Address	ETH_MAC_ ADDR	Specifies the MAC address of the node this device acts as a proxy for.	

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
12	Optional	Get	V	PRP/HSR Nodes Table Path	STRUCT of	Path to PRP/HSR Nodes Table object	
					UINT	Path Size	Number of 16 bit words in Path
					Padded EPATH	Path: Logical segments identifying the Nodes Table object	The path is restricted to one logical instance segment. The maximum size is 12 octets. See IEC 61158-6-2, 4.1.9.4
13	Required	Get/Set	NV	Switching Mode	UINT	Shows which feature is enabled in this LRE	See 7.12.4.5
14	Conditional [b]	Get/Set	NV	HSR Mode	UINT	Mode of the HSR LRE	See 7.12.4.6
15	Conditional [a]	Get/Set	NV	RedBox ID	UINT	Redundancy Box Identity	See 7.12.4.7
16	Required	Get is required Set is optional	NV	Evaluate Supervision	BOOL	Evaluate received Supervision frames	See 7.12.4.8
17	Required	Get	V	Warning LAN A	BOOL		See 7.12.4.9
18	Required	Get	V	Warning LAN B	BOOL		See 7.12.4.9
19	Required	Set	V	Warning Count LAN A	UDINT		See 7.12.4.9
20	Required	Set	V	Warning Count LAN B	UDINT		See 7.12.4.9

[a] REQUIRED if device is a Redundancy Box (RedBox)

[b] REQUIRED if device is a Node or RedBox supporting the HSR protocol

7.12.4.2 Node Type – Attribute 2

The PRP Node Type attribute identifies the operating mode of the Node. Table 152 specifies the possible values for PRP Node Type.

Table 152 – Node Type

Value	Node type
0	Deprecated (PRP Mode 0)
1	PRP Mode 1
2	HSR Mode
3 to 65 535	Reserved for future use

NOTE "Deprecated (PRP Mode 0)" is present for historical reasons. PRP existed in a version 0 (corresponding to the first edition of IEC 62439-3). After the first edition was replaced, PRP Mode 0 was deprecated and replaced by PRP Mode 1, the current version. There is a need to distinguish nodes working with version 0 and 1, because the sequence number scheme has changed between the two. This results in an incompatibility between the two versions.

7.12.4.3 LRE Interface Counters – Attribute 8

The LRE Interface Counters attribute contains counters specific to the Parallel Redundancy Protocol (PRP). These counters shall be as defined by IEC 62439-3.

All members of Attribute 8 may be reset to 0 via the Set_Attribute_Single service. Values other than 0 shall result in General Status Code 0x09 (Invalid Attribute Value) being returned.

7.12.4.4 LRE Duplicate Detection Counters – Attribute 9

The LRE Duplicate Detection Counters attribute contains counters specific to the Parallel Redundancy Protocol (PRP). These counters shall be as defined by IEC 62439-3.

All members of Attribute 9 may be reset to 0 via the Set_Attribute_Single service. Values other than 0 shall result in General Status Code 0x09 (Invalid Attribute Value) being returned.

7.12.4.5 Switching Node – Attribute 13

The Switching Node enumeration provides the ability to select a particular operation for this LRE. It is also indicated as to what Operation is valid for which Node Type. Nodes are not required to support all values. See IEC 62439-3 for more detail.

Table 153 specifies the possible values for Switching Node.

Table 153 – Switching Node

Value	Name	Description	Node Type	
			PRP Mode	HSR Mode
0	Reserved	Reserved		
1	Non-bridging node	Unspecified non-bridging node	X	X
2	Bridging unspecified	Unspecified bridging node	X	X
3	PRP node	PRP node/RedBox	X	
4	HSR RedBox SAN	HSR RedBox with regular Ethernet traffic on its interlink		X
5	HSR node	HSR switching node		X
6	HSR RedBox HSR	HSR RedBox with HSR tagged traffic on its interlink		X
7	HSR RedBox PRPa	HSR RedBox with PRP traffic for LAN A on its interlink		X
8	HSR RedBox PRPb	HSR RedBox with PRP traffic for LAN B on its interlink		X
9 to 65 535	Reserved	Reserved for future use		

7.12.4.6 HSR Mode – Attribute 14

The HSR Mode enumeration is only applicable if the LRE is an HSR bridging node or RedBox. It shows the mode of the HSR LRE. See IEC 62439-3 for more detail.

Table 154 specifies the possible values for HSR Mode.

Table 154 – HSR Mode

Value	Name	Description
0	Reserved	Reserved
1	Mode h	HSR-tagged forwarding – Default mode The HSR LRE is in mode h and bridges tagged HSR traffic
2	Mode n	No forwarding The HSR LRE is in mode n and bridging between its HSR ports is disabled. Traffic is HSR tagged.
3	Mode t	Transparent forwarding The HSR LRE is in mode t and bridges nontagged HSR traffic between its HSR ports
4	Mode u	Unicast forwarding The HSR LRE is in mode u and behaves like in mode h, except it does not remove unicast messages
5	Mode m	Mixed forwarding (HSR-tagged and non HSR-tagged) The HSR LRE is configured in mixed mode. HSR frames are handled according to mode h. Non-HSR frames are handled according to IEEE Std 802.1Q-2018 bridging rules.
6 to 65 535	Reserved	Reserved for future use

7.12.4.7 RedBox ID – Attribute 15

Applicable to RedBox HSR-PRP A and RedBox HSR-PRP B. One ID is used by one pair of RedBoxes (one configured to A and one configured to B) coupling an HSR ring to a PRP network. The integer value states the value of the path field a RedBox inserts into each frame it receives from its interlink and injects into the HSR ring. When interpreted as binary values, the LSB denotes the configuration of the RedBox (A or B), and the following 3 bits denote the identifier of a RedBox pair. See IEC 62439-3 for more detail.

Table 155 specifies the possible values for RedBox ID.

Table 155 – RedBox ID

Value	Name
0	Reserved
1	Reserved
2	Id1a
3	Id1b
4	Id2a
5	Id2b
6	Id3a
7	Id3b
8	Id4a
9	Id4b
10	Id5a
11	Id5b
12	Id6a
13	Id6b
14	Id7a
15	Id7b
16 to 65 535	Reserved for future use

7.12.4.8 Evaluate Supervision – Attribute 16

The Evaluation Supervision attribute is set to TRUE if the LRE evaluates received supervision frames. It is set to FALSE if it drops the supervision frames without evaluating.

LREs are required to send supervision frames, but reception is optional. Default value is dependent on implementation. See IEC 62439-3 for more detail.

7.12.4.9 Warnings – Attributes 17, 18, 19, 20

The warning attributes for LAN A and LAN B notify the user there is a potential problem with the PRP ports.

Attributes 17 or 18 respectively shall be asserted (1) if one or more of the following conditions exist:

1) Packet Loss Condition
   a) If LAN A does not receive either LRE frames which are HSR tagged or RCT appended or LRE supervision frames in the last 3 s but LAN B does, Attribute 17 will be asserted (1).
   b) If LAN B does not receive either LRE frames which are HSR tagged or RCT appended or LRE supervision frames in the last 3 s but LAN A does, Attribute 18 will be asserted (1).
2) Nodes Table Active Bit, Nodes Table Object Instance Attribute 3 (if implemented)
   a) If Instance Attribute 3 of the Nodes Table Object is implemented and any node in the array has Struct Member LAN B Active asserted (1) but Struct Member LAN A Active is cleared (0), Attribute 17 will be asserted (1). This will be evaluated on a 1 s basis.
   b) If Instance Attribute 3 of the Nodes Table Object is implemented and any node in the array has Struct Member LAN A Active asserted (1) but Struct Member LAN B Active is cleared (0) Attribute 18 will be asserted (1). This will be evaluated on a 1 s basis.

3) Wrong LAN Packet Counter incremented

   a) If Attribute 8 Struct member 7 "Wrong LAN A Count" has incremented in the past second, Attribute 17 will be asserted (1).

   b) If Attribute 8 Struct member 8 "Wrong LAN B Count" has incremented in the past second, Attribute 18 will be asserted (1).

Attribute 17 (Warning LAN A) and Attribute 18 (Warning LAN B), shall be cleared if none of the associated error counters are triggered within a 3 s period, as follows:

- If Attribute 17 is asserted (1) and 1a, 2a, and 3a above are false, Attribute 17 shall be cleared (0).
- If Attribute 18 is asserted (1) and 1b, 2b, and 3b above are false, Attribute 18 shall be cleared (0).

If Attribute 17 changes state from cleared (0) to asserted (1), Attribute 19 shall be incremented by 1.

If Attribute 18 changes state from cleared (0) to asserted (1), Attribute 20 shall be incremented by 1.

Attributes 19 and 20 may be reset to 0 via the Set_Attribute_Single service. Values other than 0 shall result in General Status Code 0x09 (Invalid Attribute Value) being returned.

7.12.5 Diagnostic connection points

One diagnostic connection point is defined for the PRP/HSR Protocol object class at the instance level. This connection point is required if the Standard Network Diagnostic Assembly is implemented (see IEC 61158-6-2, 4.1.8.4.1.4).

NOTE See IEC 61158-5-2, 6.1.6 for further information about Diagnostic Connection Points.

The format of the PRP/HSR Protocol diagnostic connection point is as specified in Table 156.

Table 156 – PRP/HSR Protocol connection point 1, Standard Network Diagnostics

Attribute ID	Attribute name	Data type	Attribute size	Size of structure
17	Warning LAN A	BOOL	1 bit	8 octets
18	Warning LAN B	BOOL	1 bit	
N/A	62 bits pad, shall be zeros		7 octets, 6 bits	

7.12.6 Common Services

7.12.6.1 General

The PRP/HSR Protocol object shall support the common services as specified in Table 157.

Table 157 – PRP/HSR Protocol object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Optional	Get_Attributes_All	Returns multiple attributes in numerical order
0x0E	Conditional [a]	Required	Get_Attribute_Single [d]	Returns the contents of the specified attribute or connection point
0x10	N/A	Required [c]	Set_Attribute_Single	Modifies a single attribute
0x18	N/A	Required	Get_Member [b]	Returns the content of a selected member of an attribute
0x1D	N/A	Conditional [e]	Get_Connection_Point_Member_List	Used to get member paths of a Connection Point structure

[a] The Get_Attribute_Single shall be implemented for the class attribute if any class attribute is implemented.

[b] The Get_Member service is only required to be implemented for Proxy Nodes Table (Attribute #11).

[c] Service is required but a vendor can include mechanism for the user to disable for security reasons.

[d] This service shall support the use of a connection point in the service path when Diagnostic Connection Point(s) are implemented (see 7.12.5).

[e] This service is required if Diagnostic Connection Point(s) are implemented.

7.12.6.2 Get_Attributes_All response

7.12.6.2.1 Class level

At the class level, the Get_Attributes_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

7.12.6.2.2 Instance level

The structure of the Get_Attributes_All response at the instance level is specified in Table 158.

Table 158 – Get_Attributes_All response

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	BOOL	LRE Enable	
2	UINT	Node Type	
3	SHORT_STRING	Node Name	
4	SHORT_STRING	Version Name	
5	ETH_MAC_ADDR	LRE MAC Address	
6	UINT	Duplicate Discard	
7	UINT	Transparent Reception	

Attribute ID	Data type	Attribute name	Default value (if not implemented)
8	STRUCT of	LRE Interface Counters	
	UDINT	Transmit Count A	
	UDINT	Transmit Count B	
	UDINT	Transmit Count C	
	UDINT	Receive Count A	
	UDINT	Receive Count B	
	UDINT	Receive Count C	
	UDINT	Wrong LAN A Count	
	UDINT	Wrong LAN B Count	
	UDINT	Wrong LAN C Count	
9	STRUCT of	LRE Duplicate Detection Counters	
	UDINT	Entries Unique Count A	
	UDINT	Entries Unique Count B	
	UDINT	Entries Duplicate Count A	
	UDINT	Entries Duplicate Count B	
	UDINT	Entries Multiple Count A	
	UDINT	Entries Multiple Count B	
10	UDINT	PRP Proxy Nodes Count	0
11	ARRAY of:	Proxy Nodes Table	
	ETH_MAC_ADDR	Proxy Node MAC Address	
12	STRUCT of	Path to PRP Nodes Table object	
	UINT	Path Size	0
	Padded EPATH	Path	
13	UINT	Switching Node	
14	UINT	HSR Mode	0
15	UINT	Redundancy Box Identity	0
16	BOOL	Evaluate Received Supervision Frames	
17	BOOL	Warning LAN A	
18	BOOL	Warning LAN B	
19	UDINT	Warning Count LAN A	
20	UDINT	Warning Count LAN B	

7.13 PRP/HSR Nodes Table object

7.13.1 Overview

The PRP/HSR Nodes Table object keeps the record of all PRP or HSR Capable nodes that have been detected on the network.

PRP and/or HSR capable devices participating in a PRP or HSR network may implement one or more instances of the PRP/HSR Nodes Table object.

7.13.2 Revision history

The revision history of the PRP/HSR Nodes Table object is described in Table 159.

Table 159 – Revision history

Revision	Reason for object definition update
1	Initial revision of this object definition, with PRP only (obsolete)
2	Revision of this object definition integrating HSR with PRP
3	Added conditional instance attribute 3, revised instance attribute 2 to conditional, revised Get_Attributes_All instance level service to optional

7.13.3 Class attributes

The PRP/HSR Nodes Table object shall support the class attributes as specified in Table 160.

Table 160 – Class attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	V	Revision	UINT	Revision of this object	The current value assigned to this attribute can be 2 or 3
2	Required	Get	V	Max Instance	UINT	Maximum instance number of an object currently created in this class level of the device	The largest instance number of a created object at this class hierarchy level
3	Required	Get	V	Number of Instance	UINT	Number of object instances currently created at this class level of the device	The number of object instances at this class hierarchy level
4 to 7	These class attributes are optional and are described in IEC 61158-5-2.						

7.13.4 Instance attributes

7.13.4.1 General

The PRP/HSR Nodes Table object shall support the instance attributes as specified in Table 161.

Table 161 – Instance attributes

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	V	PRP/HSR Nodes Table(s) Count	UDINT	Number of entries in the PRP/HSR Nodes Table with Time Last Seen (if implemented), number of entries in the PRP/HSR Table with Active Status (if implemented), or the number of entries in each table if both are implemented	

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
2	Conditional [a]	Get Get_ Member(s)	V	PRP/HSR Nodes Table with Time Last Seen	ARRAY of		See 7.13.4.2
					STRUCT of		
				Node MAC Address	ETH_MAC_ADDR	MAC address of the source node as advertised by the PRP Supervision frame	
				Time Last Seen A	UDINT	Time in TimeTicks (1/100 s) since the last frame from this remote LRE was received over LAN A. Initialized with a value of 0 upon node registration.	
				Time Last Seen B	UDINT	Time in TimeTicks (1/100 s) since the last frame from this remote LRE was received over LAN B. Initialized with a value of 0 upon node registration.	
				Remote Node Type	UDINT	Node type, as indicated in received Supervision frame	
3	Conditional [a]	Get Get_ Member(s)	V	PRP/HSR Nodes Table with Active Status	ARRAY of		See 7.13.4.2
					STRUCT of		
				Node MAC Address	ETH_MAC_ADDR	MAC address of the source node as advertised by the PRP Supervision frame	
				Node IP Address	UDINT	IP Address of the source node, Return 0.0.0.0 if not known.	
				LAN A Active	BOOL	Frame received in last 3 s on LAN A from this node	0 = No frame seen in last 3 s on LAN A from this node (default) 1 = Frame seen in last 3 s on LAN A from this node

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
				LAN B Active	BOOL	Frame received in last 3 s on LAN B from this node	0 = No frame seen in last 3 s on LAN B from this node (default) 1 = Frame seen in last 3 s on LAN B from this node
				Remote Node Type	UDINT	Node type, as indicated in received Supervision frame	

[a] Either Attribute 2 or Attribute 3 are required. Both may be implemented.

7.13.4.2 PRP/HSR Nodes Table with Time Last Seen and PRP/HSR Nodes Table with Active Status – Attributes 2 and 3

Either Attribute 2 or Attribute 3 are required. Both may be implemented. Attribute 2 reflects the IEC 62439-3 reference SNMP implementation, while attribute 3 is provided to reduce device resource requirements.

Since the size of the PRP/HSR Nodes Table with Time Last Seen and/or PRP/HSR Nodes Table with Active Status (Instance Attributes 2 and 3) could be large, depending on the number of nodes participating in PRP/HSR, these attributes shall be accessible with the Get_Member service (0x18), including support for Multiple Sequential Members – Extended Protocol.

Clients may elect to use the Get_Attribute_Single service to read either the PRP/HSR Nodes Table with Time Last Seen or PRP/HSR Nodes Table with Active Status. If the number of nodes participating in PRP/HSR results in a response that is too large, the PRP/HSR Object shall return error code 0x11 (Reply Data Too Large). Clients should use the Get_Member service to read individual members or the Multiple Sequential Members – Extended Protocol to read a block or blocks of members.

The Remote Node Type identifies the type of the Node, as indicated in the received Supervision frame. Table 162 specifies the possible values for Remote Node Type.

Table 162 – Remote Node Type

Value	Name	Description
0	DANP	A Double Attached Node running PRP
1	RedBoxP	A Redundancy Box running PRP
2	VDANP	A Virtual Double Attached Node running PRP
3	DANH	A Double Attached Node running HSR
4	RedBoxH	A Redundancy Box running HSR
5	VDANH	A Virtual Double Attached Node running HSR
6 to $2^{32} - 1$		Reserved for future use

7.13.5 Common services

7.13.5.1 General

The PRP/HSR Nodes Table object shall support the common services as specified in Table 163.

Table 163 – PRP/HSR Nodes Tables object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Optional	Get_Attributes_All	Returns multiple attributes in numerical order
0x0E	Required	Required	Get_Attribute_Single	Returns a single attribute
0x18	N/A	Required	Get_Member	Basic and Extended Protocol

7.13.5.2 Get_Attributes_All response

7.13.5.2.1 Class level

At the class level, the Get_Attributes_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

7.13.5.2.2 Instance level

The structure of the Get_Attributes_All response at the instance level is specified in Table 164.

Table 164 – Get_Attributes_All response

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	UDINT	PRP/HSR Nodes Table Count	
2	ARRAY of	PRP/HSR Nodes Table	
	STRUCT of	Table Entry	
	ETH_MAC_ADDR	Node MAC Address	
	UDINT	Time Last Seen A	
	UDINT	Time Last Seen B	
	UDINT	Remote NodeType	

7.14 LLDP Management object

7.14.1 Overview

The LLDP Management object contains administrative information for the LLDP protocol. Detailed requirements for the Type 2 implementation of LLDP are specified in Clause 12.

This object shall exist if LLDP is implemented on the device. Only one instance of the LLDP Management object shall be implemented.

7.14.2 Revision history

The revision history of the LLDP Management object is described in Table 165.

Table 165 – Revision history

Revision	Reason for object definition update
1	Initial revision of this object definition

7.14.3 Class attributes

The LLDP Management object shall support the class attributes as specified in Table 166.

Table 166 – Class attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1 to 7	These class attributes are optional and are described in IEC 61158-5-2.						

7.14.4 Instance attributes

7.14.4.1 General

The LLDP Management object shall support the instance attributes as specified in Table 167.

Table 167 – Instance attributes

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get/Set	NV	LLDP Enable	STRUCT of		See 7.14.4.2
				LLDP Enable Array Length	UINT	Number of bits defined in the LLDP Enable Array member of this structure.	
				LLDP Enable Array	ARRAY of octets	Enable generation of LLDP Frames both Globally and per Port and the processing of received LLDP frames globally.	
2	Required	Get/Set	NV	msgTxInterval	UINT	From IEEE Std 802.1 AB-2016. The interval in seconds at which LLDP frames are transmitted from this device	0 = Reserved 1 to 3 600 = Message Transmission Interval for LLDP frames 3 601 to 65 535 = Reserved Recommended default value is 30.
3	Required	Get/Set	NV	msgTxHold	USINT	From IEEE Std 802.1 AB-2016. A multiplier of msgTxInterval to determine the value of the TTL TLV sent to neighboring devices	0 = Reserved 1 to 100 = Message Transmission Multiplier for LLDP Frames 101 to 255 = Reserved Recommended default value is 4.

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
4	Required	Get	V	LLDP Datastore	WORD	An indication of the retrieval methods for the LLDP database supported by the device (see 12.5)	Bit: 0 = LLDP Data Table object [a] 1 = SNMP [a] 2 = NETCONF YANG 3 = RESTCONF YANG 4 to 15 = Reserved
5	Required	Get	V	Last Change	UDINT	The value of sysUpTime taken the last time any entry in the local LLDP database (ignoring TTL) changed	sysUpTime is the time (in hundredths of a second) since the network management portion of the system was last re-initialized according to IETF RFC 3418.

[a] At least one of bits 0 and 1 are required.

7.14.4.2 LLDP Enable

The LLDP Enable Array Length member indicates the number of bits defined in the LLDP Enable Array member of the LLDP Enable structure. It is equal to one plus the value of the Max Instance number (class attribute 2) of the Ethernet Link object. The value of the LLDP Enable Array Length member, divided by 8 and rounded up for any remainder, gives the length of the LLDP Enable Array member in octets.

The LLDP Enable Array member is an array of bits (grouped in octets) that will enable generation of LLDP Frames both globally and per port and the processing of received LLDP frames globally. The bit definitions of the LLDP Enable Array member are specified in Table 168.

Table 168 – Bit Definitions of the LLDP Enable Array

Bit [a]	Name	Values	Description
0	Global Enable	0 = LLDP Tx & Rx Disabled, 1 = LLDP Tx & Rx Enabled (default)	If Global Enable (Bit 0) = 0, transmitting will be stopped for all ports (other bits shall be ignored) and received LLDP frames will be ignored. On set of Global Enable (Bit 0) from 1 to 0, all existing table entries shall be removed.
1-N	Port Tx Enable	0 = LLDP Tx Disabled, 1 = LLDP Tx Enabled (default)	The values of elements corresponding to non-existent Ethernet Link instances or instances (e.g. internal) that do not support LLDP shall be set to 0 and are ignored.
> N	Reserved	Shall be 0 and are ignored.	Pad bits included as necessary to fill the last octet.

[a] The bit number of each Port Tx Enable bit corresponds to the same instance number of the Ethernet Link object, where N is the value of the Max Instance class attribute 2 of the Ethernet Link object.

7.14.5 Common services

7.14.5.1 General

The LLDP Management object shall support the common services as specified in Table 169.

Table 169 – LLDP Management object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Optional	Get_Attributes_All	Returns multiple attributes in numerical order
0x0E	Conditional [a]	Required	Get_Attribute_Single	Returns a single attribute
0x10	N/A	Required	Set_Attribute_Single	Modifies a single attribute

[a] Required if class attributes are implemented

7.14.5.2 Get_Attributes_All response

7.14.5.2.1 Class level

At the class level, the Get_Attributes_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

7.14.5.2.2 Instance level

The structure of the Get_Attributes_All response at the instance level is specified in Table 170.

Table 170 – Get_Attributes_All response

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	STRUCT of	LLDP Enable	
	UINT	LLDP Enable Array Length	
	ARRAY of octets	LLDP Enable Array	
2	UINT	msgTxInterval	
3	USINT	msgTxHold	
4	WORD	LLDP Datastore	
5	UDINT	Last Change	

7.15 LLDP Data Table object

7.15.1 Overview

The LLDP Data Table object displays a record of all adjacent LLDP implementing devices that are currently active according to the receive state machine of the LLDP protocol. Detailed requirements for the Type 2 implementation of LLDP are specified in Clause 12.

This object shall exist if LLDP is implemented on the device and the SNMP LLDP MIB has not been implemented. If neighboring devices have not been detected, only class attributes can be reachable.

One instance of the LLDP Data Table object shall be implemented per adjacent device detected. At least 8 instances shall be supported. It is recommended that 48 instances are supported. Instances shall be created and removed as neighboring devices change. The same instance number should be maintained for each neighboring device until the next power cycle of the device implementing this object.

7.15.2 Revision history

The revision history of the LLDP Management object is described in Table 171.

Table 171 – Revision history

Revision	Reason for object definition update
1	Initial revision of this object definition

7.15.3 Class attributes

The LLDP Management object shall support the class attributes as specified in Table 172.

Table 172 – Class attributes

Attribute ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	This class attribute is optional and is described in IEC 61158-5-2.						
2	Required	Get	V	Max Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
3	Required	Get	V	Number of Instances	UINT	Number of object instances currently created at this class level of the device.	The number of object instances at this class hierarchy level
4 to 7	These class attributes are optional and are described in IEC 61158-5-2.						

7.15.4 Instance attributes

7.15.4.1 General

The LLDP Management object shall support the instance attributes as specified in Table 173.

Table 173 – Instance attributes

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
1	Required	Get	V	Ethernet Link Instance Number	UINT	Local instance number of the Ethernet Link Object that matches the physical Ethernet port the LLDP frame populating this instance was received on, if known.	0 = Unknown 1 to 65 535 = Ethernet Link Object instance number

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
2	Required	Get	V	MAC Address	ETH_MAC_ADDR	Neighboring MAC Address received from the Type 2 MAC Address, Chassis ID, or Port ID TLV	The MAC address will be set by the first occurrence of a MAC ID that exists in the following list: 1.Type 2 MAC Address (TLV Type = 127, subtype = 2) 2. Chassis ID (TLV Type = 1) only if subtype = 2 3. Port ID (TLV Type = 2) only if subtype = 3 4. All zero
3	Required	Get	V	Interface Label	SHORT_STRING	Neighboring Interface Label received from the Type 2 Interface Label, Chassis ID or Port ID TLV	The Interface Label will be a maximum of 64 characters. It is set by the first occurrence of an interface label that exists in the following list: 1. Type 2 Interface Label (TLV Type = 127, Subtype = 1) 2. Chassis ID (TLV Type = 1) only if subtype = 6 3. Port ID (TLV Type = 2) only if subtype = 5 4. A null string
4	Required	Get	V	Time to Live	UINT	Number of seconds the neighboring information is to be considered valid.	(TLV Type = 3) 0 = Reserved 1 to 65 535 = Time To Live (in seconds) NOTE A received TTL TLV value of 0 means the table entry should be removed per IEEE Std 802.1AB-2016.
5	Required	Get	V	System Capabilities TLV	STRUCT of	Contain bitmaps of the capabilities that define the primary function(s) of the neighboring system.	(TLV type = 7) See 7.14.4.2
				System Capabilities	WORD	Capabilities which the neighboring device supports based on currently loaded firmware.	
				Enabled Capabilities	WORD	Capabilities currently enabled on the neighboring device.	

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
6	Required	Get	V	IPv4 Management Addresses	STRUCT of	IPv4 management addresses of the neighboring device	List of IPv4 encoded management addresses as defined by one or more received Management Address TLVs (TLV Type = 8) Exclude non-IPv4 Management Addresses.
				Management Address Count	USINT	Number of implemented management addresses	0 to 255 = Number of received Management Address TLV's from this neighbor.
				Management Address	ARRAY of UDINT	IPv4 Management Addresses of the neighboring device	The IP address shall be set to a valid Class A, B, or C address and shall not be set to all zeros or the loopback address (127.0.0.1).
7	Required	Get	V	Type 2 Identification	STRUCT of	Type 2 Identification TLV of the neighboring device, if present.	Set by the Type 2 Identification TLV (TLV Type = 127, Subtype = 01), if present. Otherwise 0 See IEC 61158-6-2, 4.1.9.4.2, Key format 5, for details and semantics
				Vendor ID	UINT		
				Device Type	UINT		
				Product Code	UINT		
				Major Revision	SWORD		
				Minor Revision	USINT		
				Type 2 Serial Number	UDINT		

Attr ID	Need in implementation	Access rule	NV	Name	Data type	Description of attribute	Semantics of values
8	Required	Get	V	Additional Ethernet Capabilities	STRUCT of	TLV for Ethernet preemption Support from the neighboring device	Set by the Additional Ethernet Capabilities TLV (TLV type = 127, Subtype 7), if present. Otherwise 0 See IEEE Std 802.3-2018, 79.3 for more information
				Preemption Support	BOOL	Allows a link partner to know if preemption is supported on the link	0 = Not supported 1 = Supported
				Preemption Status	BOOL	Allows a link partner to know if preemption is enabled on the link	0 = Not enabled 1 = Enabled
				Preemption Active	BOOL	Allows a link partner to know if link preemption has passed embedded verification	0 = Not active 1 = Active
				Additional Fragment Size	USINT	Number of octets that shall be transmitted of a frame before preemption occurs	0 = 64 octets 1 = 128 octets 2 = 192 octets 3 = 256 octets 4 to 255 = Reserved
9	Required	Get	V	Last Change	UDINT	The value of sysUpTime taken the last time any attribute in this instance changed	sysUpTime is the time (in hundredths of a second) since the network management portion of the system was last re-initialized per IETF RFC 3418.

7.15.4.2 System Capabilities TLV

The System Capabilities TLV attribute is a structure that contains bitmaps of both the supported and enabled capabilities of the neighboring device. The structure of these bitmaps is specified in Table 174.

Table 174 – Bitmaps of supported capabilities & enabled capabilities

Bit	Definition
0	Other
1	Repeater
2	Bridge
3	Access Point
4	Router
5	Telephone [a]
6	DOCSIS Cable Device [a]
7	End Station
8	C-VLAN component [a]
9	S-VLAN component [a]
10	Two-port MAC Relay Component [a]
11 – 15	Reserved by IEEE
[a] Bits 5, 6, and 8 to 11 are defined in the IEEE Specification but are unlikely in Type 2 networks	

One or more capabilities may be supported.

See IEEE Std 802.1AB-2016, 8.5.8 for more details. Type 2 Ethernet Bridged multiport neighboring devices are expected to assert high bits 0 and 2 as specified in 12.3.4.

7.15.5 Common services

7.15.5.1 General

The LLDP Management object shall support the common services as specified in Table 175.

Table 175 – LLDP Management object common services

Service code	Need in implementation		Service name	Description of service
	Class	Instance		
0x01	Optional	Optional	Get_Attributes_All	Returns multiple attributes in numerical order
0x0E	Required	Required	Get_Attribute_Single	Returns a single attribute
0x11	Required	N/A	Find_Next_Object_Instance	Causes the specified class to search for and return a list of instance IDs of existing instances of the LLDP Data Table object.

7.15.5.2 Get_Attributes_All response

7.15.5.2.1 Class level

At the class level, the Get_Attributes_All response shall contain the class attributes in numerical order, up to the last implemented attribute. Any unimplemented attributes in the response shall use the default attribute values.

7.15.5.2.2 Instance level

The structure of the Get_Attributes_All response at the instance level is specified in Table 176.

Table 176 – Get_Attributes_All response

Attribute ID	Data type	Attribute name	Default value (if not implemented)
1	UINT	Ethernet Link Instance Number	
2	ETH_MAC_ADDR	MAC Address	
3	SHORT_STRING	Interface Label	
4	UINT	Time to Live	
5	STRUCT of	System Capabilities TLV	
	WORD	System Capabilities	
	WORD	Enabled Capabilities	
6	STRUCT of	IPv4 Management Addresses	
	USINT	Management Address Count	
	ARRAY of UDINT	Management Address	
7	STRUCT of	Type 2 Identification	
	UINT	Vendor ID	
	UINT	Device Type	
	UINT	Product Code	
	SWORD	Major Revision / Compatibility	
	USINT	Minor Revision	
	UDINT	Serial Number	
8	STRUCT of	Additional Ethernet Capabilities	
	BOOL	Preemption Support	
	BOOL	Preemption Status	
	BOOL	Preemption Active	
	USINT	Additional Fragment Size	
9	UDINT	Last Change	

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 177 and Figure 27.

Table 177 – 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 ms – 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"

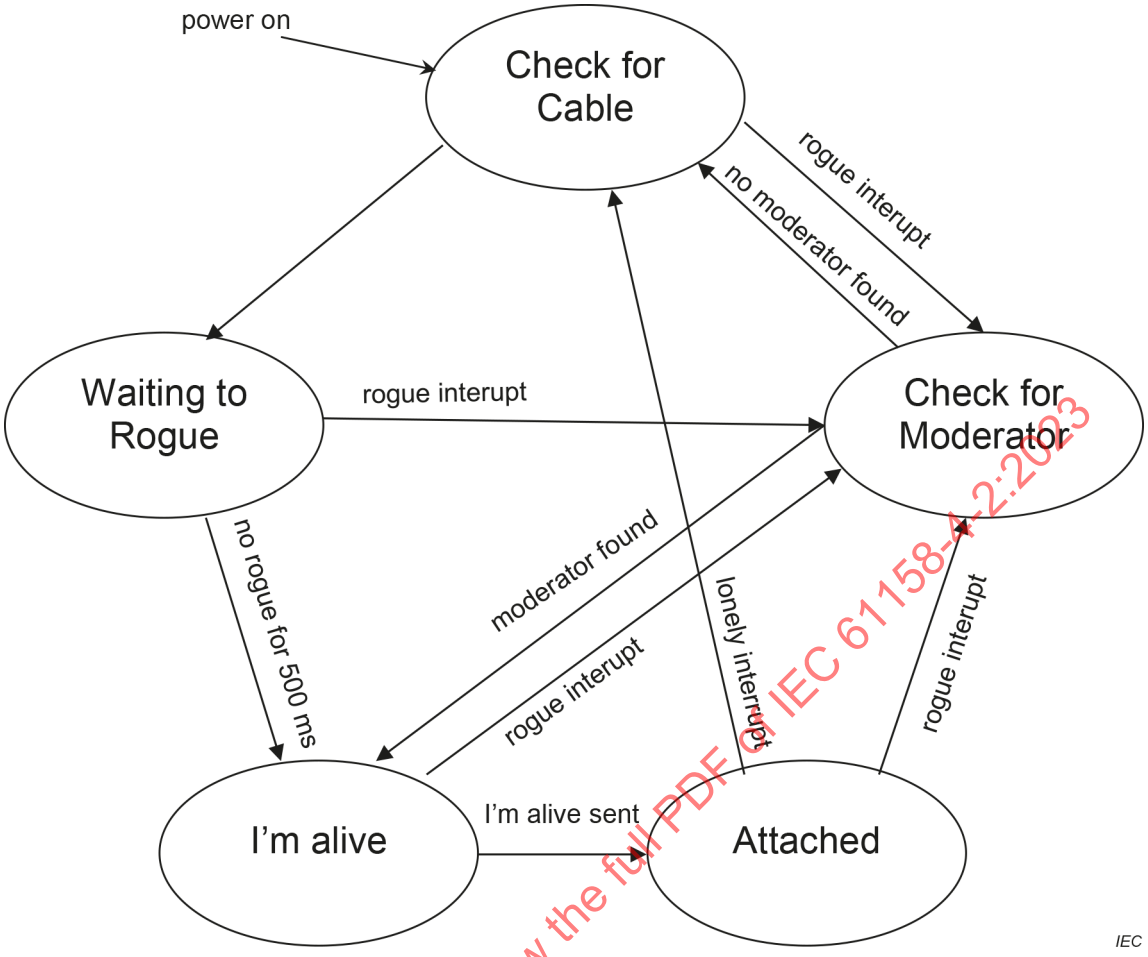


Figure 27 – NAM state machine

8.1.2 Default parameters

The default link parameters for a node shall be as shown in Table 178:

Table 178 – 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_recv_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,
            {
            count = preset;     // start with 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_recv_fixed_indication
              || DLL_recv_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);

    event:    MAC has not transmitted any frames for 1300 to 1500 msec
    destination: checkForCable0
    action:
       DLL_set_current_request(default_config);      // set default parameters


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_recv_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_recv_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_recv_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_recv_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_recv_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: check ForModerator3
    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

Subclause 8.2 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 could be possible that the new moderator received the last moderator DLPDU from the previous moderator, while other nodes would 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 8.2.4 assume a perfectly accurate clock. Any permitted inaccuracy is explicitly included in the requirements. Sources of inaccuracy can 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 Signaling to support the tone and NUT precision shall be as defined in Table 179.

Table 179 – PhL timing characteristics

Specification	Limits / characteristics	Comments
Data Rate	5 Mbit/s ± CA	Also called M_Symbol rate, *data "zero" or "one"*
Bit Time	200 ns ± CA	Also called M_Symbol time, *data "zero" or "one"*
PhL symbol time	100 ns ± CA	Also called Phy_Symbol time, see data encoding rules
Clock Accuracy (CA)	± 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 (`NUT_length` × 10 µs) ± 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 (`NUT_length` – `gb_center`) × 10 µs ± CA after the tone.

8.2.4.5 Non-moderator node timing

A non-moderator node shall begin a new NUT at (`gb_center` × 10 − 40 ± 2) µs 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 (`NUT_length` × 10) µs ± 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 (tone + 20) µs and (tone + (`NUT_length` × 10 − 30) µs ± CA).

8.2.4.6 Post-tone silence

No DLPDUs shall be transmitted between the tone and 20 µs after the tone.

8.2.4.7 Pre-tone silence

No DLPDU shall be transmitted, except the moderator DLPDU, that does not terminate before ((NUT_length − gb_prestart) × 10 + 11,2) µs ± 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 (NUT_length − gb_start) × 10 µs ± 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 ((NUT_length − gb_start) × 10 + 20) µs ± 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 (NUT_length × 10 − 40) µs ± 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 (tone + 20 µs) and (tone + (NUT_length − gb_start) × 10 µs ± 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 slotTime × 1 µs ± 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.

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

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 <stddef.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 nuttenth;
   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 [-abdhlnrt] [-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 = atol(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 is 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 shall 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 , 1000000L);
```

```
        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

Clause 9 specifies each main functional component of the DL internal architecture as shown in Figure 1 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  nqlowman;     // 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  macSrce;      // 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_overrun,
   DLL_EV_dribble,
   DLL_EV_nonconcurrence,
   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 mac_ID);
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_nonconcurrence);   // indicate non-concurrence
       }
   else
       {
       if(!in_guardband)            // unexpected moderator
           {
           DLL_event(DLL_EV_nonconcurrence);   // indicate non-concurrence
           }

       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 (nqlowman == 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 = 255;
    else
        modcountinit =TXDUPCHECKS;

    // 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,macSrce);
        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,macSrce);
        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_overrun);
          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_rxAbort);
        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_nonconcurrence);
        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 && fifo[SCHEDULED].has_data())
            {
            DLL_event(DLL_EV_dribble);
            }

//
// send subsequent Lpackets
//
state: sendNextLpacket

    // last Lpacket was aborted, end frame
    // note that the TXM has already sent the abort sequence on the wire
    event:       TX_LpacketComplete
    condition:   Tx_abort==TRUE
    destination: endFrame
    action:
        DLL_event(DLL_EV_txAbort);
        gen_timer.start(5,2 usec);
        // start hold timer after line is quiet
        holdoff_timer.start((DLL_SM_current.blanking+1) * 1,6 usec);

    // TxLLC has something, send a new Lpacket
    event:    TX_LpacketComplete
    condition:   (TX_abort == FALSE) && (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:   (TX_abort == FALSE)&&(LLC_pickLpacket(scheduled,octetsleft) == FALSE)
    destination: sendTrailer
    action:
        TX_sendTrailer();

//
// complete DLPDU termination
//
state: sendTrailer

    event:    TX_trailerComplete
    destination: endFrame
    action:
        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_req)
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_req)
       && 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_req)
           && 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_req)
           && 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.ctl != 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.ctl == 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.ctl == MODERATOR_CTL
        && RX_Lpacket.service == MODERATOR_TAG
        && RX_Lpacket.dest == 0xFF
        && macSrce <= qlowman
        && (macSrce == 0 || DLL_SM_current.myaddr == 0xFF)
destination: waitTone
action:
    qlowman = min(qlowman, 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.ctl == MODERATOR_CTL
        && RX_Lpacket.service == MODERATOR_TAG
        && RX_Lpacket.dest == 0xFF
        && macSrce <= qlowman
        && !(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:
    qlowman = min(qlowman, 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.ctl == MODERATOR_CTL
        && RX_Lpacket.service == MODERATOR_TAG
```

```
            && RX_Lpacket.dest == 0xFF
            && macSrce <= qlowman
            && !(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:
            qlowman = min(qlowman, 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.ctl == MODERATOR_CTL
            && RX_Lpacket.service == MODERATOR_TAG
            && RX_Lpacket.dest == 0xFF
            && macSrce <= qlowman
            && !(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.ctl = 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 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
//
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_recv_fixed_indication (
          USINT Lpacket[],
          UINT  size,
          USINT service,
          USINT sourceID);

DLL_recv_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
    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);

    // init for a new Lpacket
    void init(void);
    };


class rxLpacket: public Lpacket
    {
    public:
    USINT  source;       // the MAC ID of the node that sent this Lpacket
    intmemory_size; // size of the Lpacket in memory
    BOOL   fixed;    // true if fixed , false if generic
    };


////////////////////////////////////////////////////////////////
//
// 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];
        }
    };


////////////////////////////////////////////////////////////////////////
//
// the interface to the tag filter lookup tables -- the implementation is unspecified
//
class gen_screener
```

```
    {
    public:
    BOOL enable(gen_tag tag); // set tag, requires tag, TRUE=success/FALSE=failure
    BOOL disable(gen_tag tag);// clear tag, requires tag, TRUE=success/FALSE=failure
    BOOL lookup(gen_tag tag); // lookup tag, requires tag, TRUE=enabled/FALSE=disabled
    void init(void);    // powerup init & clear
    };

class fixed_screener
    {
    public:
    BOOL enable(USINT service);   // set tag, requires tag, TRUE=success/FALSE=failure
    BOOL disable(USINT service);  // clear tag, requires tag, TRUE=success/FALSE=failure
    BOOL lookup(USINT service);   // lookup tag, requires tag,
TRUE=enabled/FALSE=disabled
    void init(void);    // powerup init & clear
    };


////////////////////////////////////////////////////////////////////////////
//
// 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 checking shall be the same as specified for HDLC (ISO/IEC 13239), 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 ACM_tMinus;            // 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;
    }
```

10 Device Level Ring (DLR) protocol

10.1 General

Clause 10 defines the Device Level Ring (DLR) protocol, a Layer 2 protocol that provides media redundancy in a ring topology. The DLR protocol is intended primarily for implementation in Type 2 Ethernet end devices that have multiple Ethernet ports and embedded switch technology. The DLR protocol provides for fast network fault detection and reconfiguration in order to support the most demanding control applications.

Since the DLR protocol operates at Layer 2 (in the OSI network model), the presence of the ring topology and the operation of the DLR protocol are transparent to higher layer protocols such as TCP/IP and Type 2 Ethernet, with the exception of a DLR object that provides a configuration and diagnostic interface for Type 2 Ethernet.

A DLR network includes at least one node configured to be a ring supervisor, and any number of normal ring nodes. It is assumed that all the ring nodes have at least two Ethernet ports and incorporate embedded switch technology.

Non-DLR multi-port devices – switches or end devices – may be placed in the ring, subject to certain implementation constraints (e.g., no MAC table filtering). Non-DLR devices will also impact the worst-case ring recovery time.

The DLR protocol definition includes a number of aspects:

- a set of end node behaviors, for ring supervisors and normal ring nodes;
- protocol messages and associated state diagrams;
- implementation requirements for devices.

10.2 Support for Multiple DLR Ring Pairs

Devices may have multiple pairs of ring ports; Figure 28 depicts a device with 4 ring port pairs. While not depicted, a device may also have additional ports that are not DLR capable.

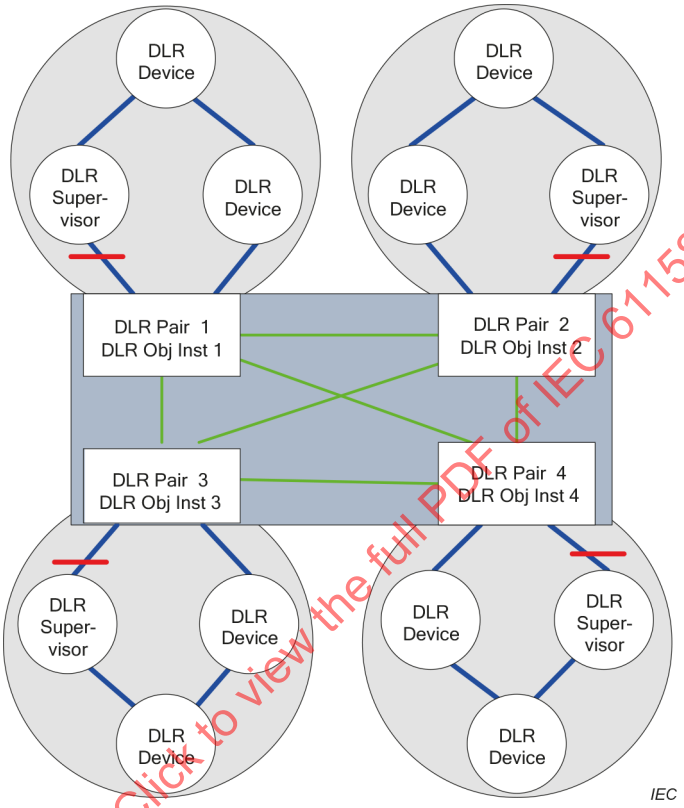


Figure 28 – Devices with Multiple DLR Ring Pairs

The DLR specification (i.e. this Clause 10) is written assuming the context of a DLR device that has only two external ports, i.e. one pair of DLR ports.

On devices that support multiple pairs of DLR ports, each pair of DLR ports shall operate independently from the other.

Each pair of ring ports shall logically have its own state machine(s), each of which has their own set of DLR parameters (many of which are reflected through the associated DLR object instance), MAC address tables, etc. The independent pairs may internally share physical resources provided that independent operation is realized. For example, a device may have only one (internally global) unicast MAC table. In this case, when the specification requires that the unicast MAC table be flushed, only the addresses for nodes on the associated ring are required to be flushed from a shared table.

Any functionality associated with internal connectivity between its ring pairs (as indicated by the green lines in Figure 28) that a multi-pair device chooses to support is beyond the scope of this document.

10.3 Supported topologies

The DLR protocol supports a simple, single-ring topology; it has no concept of multiple or overlapping rings. A network installation may however use more than one DLR-based ring, so long as each of the rings are isolated such that DLR protocol messages from one ring are not present on another ring.

The DLR protocol may coexist with, but does not interface with, standard network protocols such as IEEE Spanning Tree Protocols (STP, RSTP, MSTP), and also with vendor-specific redundancy protocols. That is, users may construct network topologies with DLR protocol rings connected to switches that are running Spanning Tree or other ring protocols, as shown in Figure 29.

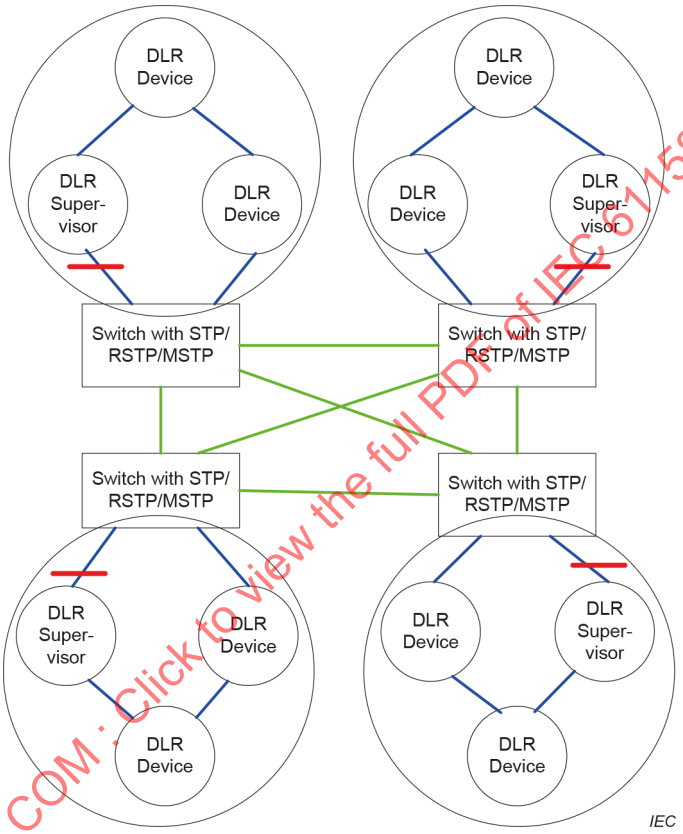


Figure 29 – DLR rings connected to switches

In Figure 29, each DLR ring is a separate DLR network, each with a ring supervisor. The supervisors are shown with one port in blocked mode, which is the case when there are no faults in the ring.

The switches to which the DLR rings are connected may run STP/RSTP/MSTP to prevent loop free operation when redundant paths are present (indicated by the green lines in Figure 29). Spanning Tree Protocol messages (BPDUs) that are sent by the switch on the DLR ring ports will be blocked by the DLR Ring Supervisor, so that the switches do not block the DLR ports (see IEEE Std 802.1Q-2018 STP/RSTP/MSTP considerations in 10.7.4).

The switches' ports to which the DLR devices are connected shall be configured properly in order ensure proper functioning of the network (see 7.9).

More complicated topologies combining DLR rings and non-DLR switches running STP/RSTP/MSTP could result in DLR ports being blocked in an undesirable manner. See 7.9 for additional information.

DLR supports redundant gateways for connecting with network infrastructure outside of the DLR network to the DLR network itself. See 10.9 for additional information.

10.4 Overview of DLR operation

10.4.1 Normal operation

Figure 30 shows the normal operation of a DLR network.

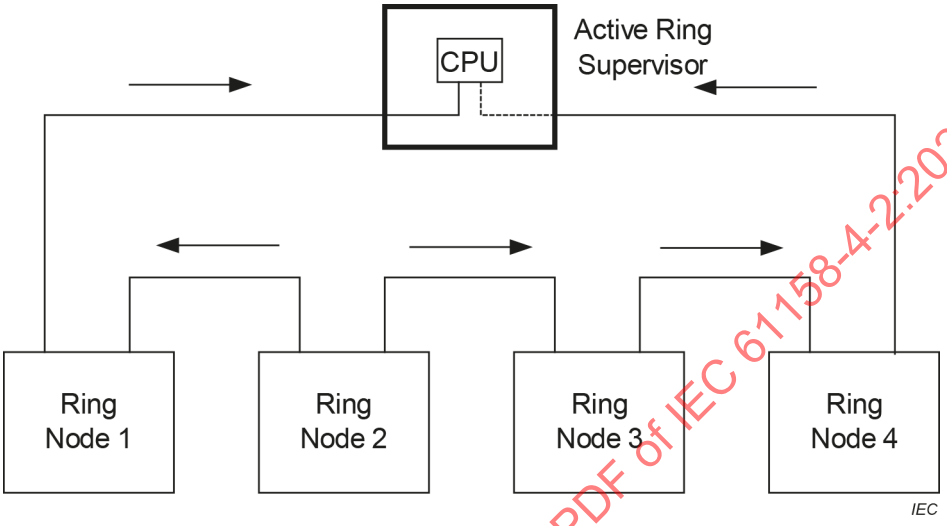


Figure 30 – Normal operation of a DLR network

and is assumed to have implemented an embedded switch. When a ring node receives a packet on one of its Ethernet ports, it determines whether the packet needs to be received by the ring node itself (e.g., the packet has the node's MAC address) or whether the packet shall be sent out the node's other Ethernet port.

The active ring supervisor blocks traffic on one of its ports with the exception of few special frames and does not forward traffic from one port to other. Because of this configuration a network loop is avoided and only one path exists between any two ring nodes during normal operation.

Figure 31 illustrates the use of Beacon and Announce frames sent by the active ring supervisor.

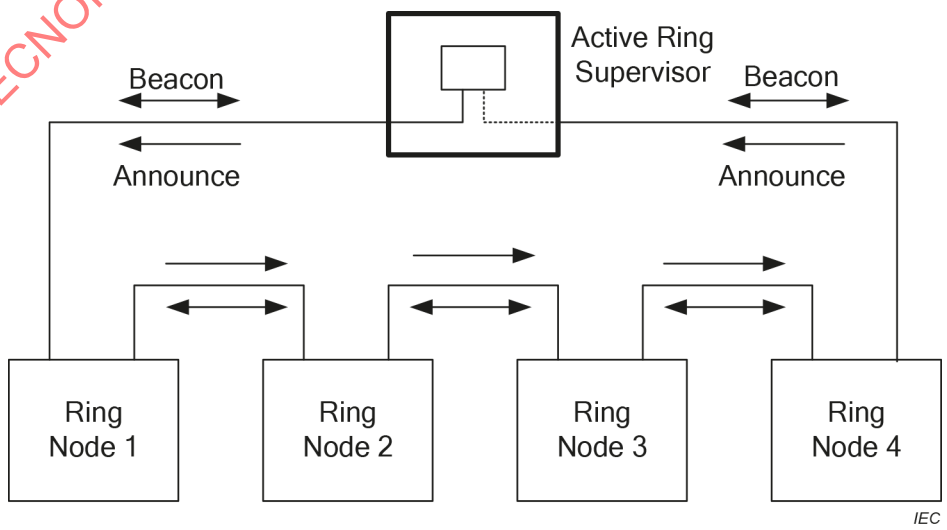


Figure 31 – Beacon and Announce frames

The active ring supervisor transmits a Beacon frame through both of its Ethernet ports once per beacon interval (400 µs by default). The supervisor also sends Announce frames once per second. The Beacon and Announce frames serve several purposes:

a) the presence of Beacon and Announce frames inform ring nodes to transition from linear topology mode to ring topology mode;

b) loss of Beacon frames at the supervisor enables detection of certain types of ring faults. (normal ring nodes are also able to detect and signal ring faults);

c) the Beacon frames carry a precedence value, allowing selection of an active supervisor when multiple ring supervisors are configured.

10.4.2 Link failures

10.4.2.1 Common failures

The most common form of link failure includes the following cases:

a) link or other physical layer failure recognized by a node adjacent to the failure;

b) power failure or power cycling a ring node, recognized by the adjacent node as a link failure;

c) intentional media disconnect by user to bring new nodes online or to remove existing ones.

In the above cases, the nodes adjacent to the fault send a Link_Status message to the ring supervisor. Figure 32 shows ring nodes adjacent to a fault sending a Link_Status message to the ring supervisor.

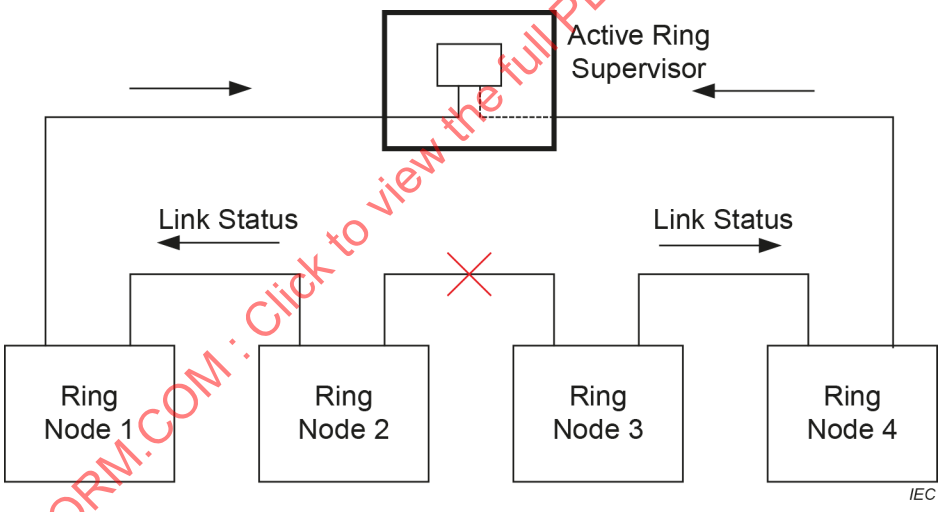


Figure 32 – Link failure

After receipt of the Link_Status message, the ring supervisor reconfigures the network by unblocking traffic on its previously blocked port and flushing its unicast MAC table. The supervisor immediately sends Beacon and Announce frames with the ring state value indicating that the ring is now faulted.

Ring nodes also flush their unicast MAC tables upon detecting loss of the beacon in one direction, or upon receipt of Beacon or Announce frames with the ring state value indicating the ring fault state. Flushing the unicast MAC tables at both supervisor and ring nodes is necessary for network traffic to reach its intended destination after the network reconfiguration.

Figure 33 shows the network configuration after a link failure, with the ring supervisor passing traffic through both of its ports.

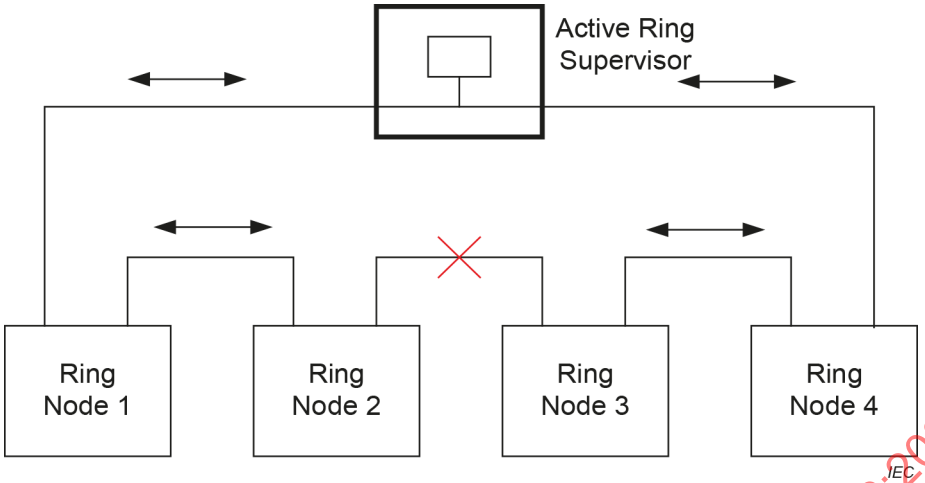


Figure 33 – Network reconfiguration after link failure

10.4.2.2 Uncommon failures

In addition to the more common link failures, there is a class of uncommon failures:

a) higher level hardware/firmware component(s) on a ring node has failed leading to lost traffic, but the physical layer is functioning normally with power supply intact;

b) a chain of ring protocol unaware nodes are connected between protocol-aware nodes, and the failure has occurred somewhere in the middle of this chain.

In these cases, the ring supervisor will detect the loss of Beacon frames first on one port, and eventually on both of its ports. The supervisor will reconfigure the network as described in the 10.4.2.1. In addition, the ring supervisor will send a Locate_Fault frame to diagnose the fault location (see 10.6.5 on the Neighbor Check process).

10.4.2.3 Partial network fault

It is possible for a partial network fault to occur such that traffic is lost in only one direction. The active ring supervisor detects a partial fault by monitoring the loss of Beacon frames on one port. When a partial fault is detected the active supervisor blocks traffic on one port and sets a status value in the DLR Object. The ring at this point will be segmented due to the partial fault, requiring user intervention.

10.4.2.4 Rapid fault/restore cycles

Certain conditions such as a faulty network connector can cause the ring supervisor to detect a series of rapid fault/restore cycles. If left to persist, such a condition could result in network instability that is difficult to diagnose. When the active supervisor detects the rapid fault/restore condition (5 faults in a 30 s period), it sets a status value in the DLR object, and blocks traffic on one port. The user shall explicitly clear the condition via the DLR object.

10.5 Classes of DLR implementation

There are several classes of DLR implementation, as described below. Detailed requirements for each class of implementation are further specified in 10.6.

- Ring Supervisor

  This class of device is capable of being a ring supervisor. Such devices shall implement the required ring supervisor behaviors, including the ability to send and process Beacon frames at the default beacon interval of 400 µs. Smaller beacon intervals, as low as 100 µs, may be supported, but are not required.

- Ring Node, Beacon-based

This class of device implements the DLR protocol, but without the ring supervisor capability. The device shall be able to process and act on the Beacon frames sent by the ring supervisor. Beacon-based ring nodes shall support beacon rates of 100 µs to 100 ms in order to accommodate all ring supervisor implementations.

- Ring Node, Announce-based

  This class of device implements the DLR protocol, but without the ring supervisor capability. In order to accommodate nodes that do not have the capacity to process Beacon frames, ring nodes may simply forward, but need not explicitly process, Beacon frames. Such nodes shall process Announce frames.

10.6 DLR behavior

10.6.1 DLR variables

Table 180 summarizes variables used in the DLR protocol behavior and messages. See 10.6.3 and 10.6.2 on Ring Node and Ring Supervisor behavior and DLR messages for further details. The DLR object (see 7.9) exposes these variables (with the exception of the Node State) via object attributes.

Table 180 – DLR variables

DLR variable	Description
Node State	Internal state of a node's DLR state machine: IDLE_STATE – initial state for non-supervisors, indicating linear topology mode FAULT_STATE – initial state for enabled ring supervisor, or when ring fault has been detected (both supervisor and ring nodes) NORMAL_STATE – normal function in ring topology mode
Ring State	State of the ring network, transmitted by ring supervisors in Beacon and Announce frames: RING_NORMAL_STATE – Ring is functioning, with supervisor blocking traffic on one port RING_FAULT_STATE – Fault detected, ring supervisor is not blocking traffic (also is the initial state transmitted in the Beacon and Announce frames)
Beacon Interval	Interval at which the ring supervisor sends Beacon frames. Supervisors shall support a range from 400 µs to 100 ms. The default value shall be 400 µs. Supervisors may support a Beacon Interval smaller than 400 µs, but this is not required. The absolute minimum Beacon Interval is 100 µs. Beacon-based ring nodes shall support beacon rates of 100 µs to 100 ms in order to accommodate all ring supervisor implementations
Beacon Timeout	Amount of time nodes shall wait before timing out reception of Beacon frames and taking the appropriate action (depending on whether supervisor or normal ring node). Supervisors shall support a range from 800 µs to 500 ms. The default shall be 1 960 µs. Supervisors may support a Beacon Timeout of smaller than 800 µs but this is not required. The absolute minimum Beacon Timeout is 200 µs
Supervisor Precedence	Precedence value assigned to a ring supervisor, and transmitted in Beacon frames. Used to select active ring supervisor when multiple supervisors have been configured. Default value is 0. Can be changed via the DLR object
DLR VLAN ID	VLAN ID used when sending DLR protocol frames. The VLAN ID is configured at the ring supervisor (via the DLR object), and is then detected by ring nodes when they receive and process the Beacon or Announce frames from the supervisor. Default value is 0. Typically the VLAN ID does not need to be changed unless a commercial switch is being used in the ring

10.6.2 Ring supervisor

10.6.2.1 Startup

An enabled ring supervisor shall start in FAULT_STATE and configure both ports to forward frames. The supervisor shall send Beacon frames out both of its ports, with the Ring State set

to RING_FAULT_STATE. The supervisor shall also send Announce frames out both of its ports with the Ring State set to RING_FAULT_STATE.

Once the Beacon frames are received through both ports the supervisor shall transition to NORMAL_STATE, flush its unicast MAC address table and reconfigure one of its ports not to forward packets, except for the following, which shall be forwarded to the host for processing:

- Beacon frames with the supervisor's own MAC address (in general needed only for software implementations);
- Beacon frames from other ring supervisors;
- Link_Status/Neighbor_Status frames;
- Neighbor_Check request or response, and Sign_On: always forward received frames. For frames originated by the supervisor, only forward frames with the Source Port matching the blocked port.

Upon transition to NORMAL_STATE, the Ring State in the Beacon frames shall be set to RING_NORMAL_STATE. The ring supervisor shall also send an Announce frame out one port, with Ring State set to RING_NORMAL_STATE.

10.6.2.2 Multiple ring supervisors

When multiple ring supervisors are configured, each supervisor sends Beacon frames when it comes online. The Beacon frames carry a supervisor precedence value. When a supervisor receives a Beacon frame, it checks the precedence value. If the precedence in the Beacon frame is higher than the receiving node's precedence value, the receiving node transitions to FAULT_STATE and becomes a backup supervisor. If the precedence values are the same, the node with the numerically higher MAC address becomes the active supervisor.

The backup supervisors configure their DLR parameters with the values obtained from the active supervisor's Beacon frames: Beacon Interval, Beacon Timeout, VLAN ID.

The backup supervisors continue to monitor both ports for timeout of the Beacon frames (no Beacons received within the Beacon Timeout period). If the Beacon has timed out on both ports, the backup supervisor waits for an additional Beacon Timeout period (during which time other nodes transition to linear mode), then begins sending its own Beacons so that a new supervisor can be selected.

10.6.2.3 Sign on

In order to identify ring protocol participants, the active ring supervisor shall send a Sign_On frame when it transitions to NORMAL_STATE. See the Sign_On information in the DLR Messages (see 10.10.9).

10.6.2.4 Normal ring operation

When in the NORMAL_STATE, the active ring supervisor shall send Beacon frames out both of its ports. It shall also send an Announce frame once per second out one port.

One of the active supervisor's ports shall be configured not to forward frames, with the exceptions as noted in 10.4.2.1.

10.6.2.5 Ring fault detection

One of several possible events shall cause the active ring supervisor to transition to FAULT_STATE:

a) Beacon frame received from another supervisor with a higher precedence value;

b) loss of Beacon frames on either port for the period specified by the Beacon Timeout, indicating a break somewhere in the ring;

c) detection of loss of link with the neighboring node on either port;

d) Link_Status frame received from a ring node, indicating a ring node has detected a fault.

In all of the cases listed above, the active ring supervisor shall:

- transition to FAULT_STATE;
- flush its unicast MAC address table;
- unblock the blocked port;
- send Beacon frame out both ports, with Ring State set to RING_FAULT_STATE;
- send Announce frame out both ports, with Ring State set to RING_FAULT_STATE.

In addition, in case 2 above, the active ring supervisor shall initiate the Neighbor Check process by issuing a Locate Fault frame. The supervisor shall also issue its own Neighbor Check frame through the port(s) on which the beacon has timed out.

When in FAULT_STATE the ring supervisor shall continue to send Beacon frames, in order to detect ring restoration.

10.6.2.6 Ring restoration

When the active ring supervisor is in FAULT_STATE, receipt of Beacon frames on both ports shall cause a transition to NORMAL_STATE. The active ring supervisor shall do the following:

- flush the unicast MAC address table;
- reconfigure its ports such that traffic is not forwarded on one port (with exceptions as noted previously);
- send Beacon frames with the Ring State set to RING_NORMAL_STATE;
- send Announce frames out one port with Ring State set to RING_NORMAL_STATE.

10.6.2.7 Changing ring parameters

The following ring supervisor parameters may be changed via the DLR object (see 7.9):

- Supervisor precedence value;
- Beacon interval;
- Beacon timeout;
- VLAN ID;
- Supervisor enabled/disabled.

When any of the above parameters are changed on the active ring supervisor, the ring supervisor shall cease sending Beacon and Announce frames for two (current) beacon timeout periods, then shall send Beacon frames using the new parameters. Ceasing the Beacon frames allows beacon-based ring nodes to detect the new parameters when the Beacon is restored and triggers active supervisor negotiation based on the new parameters. Announce frame production is further suppressed for at least 2 new beacon timeout periods to make sure that the active supervisor is determined before any Announce frames with the new parameters are sent.

When parameters are changed on a backup ring supervisor, the behavior depends on the backup supervisor's new precedence value compared to the active supervisor's precedence value: