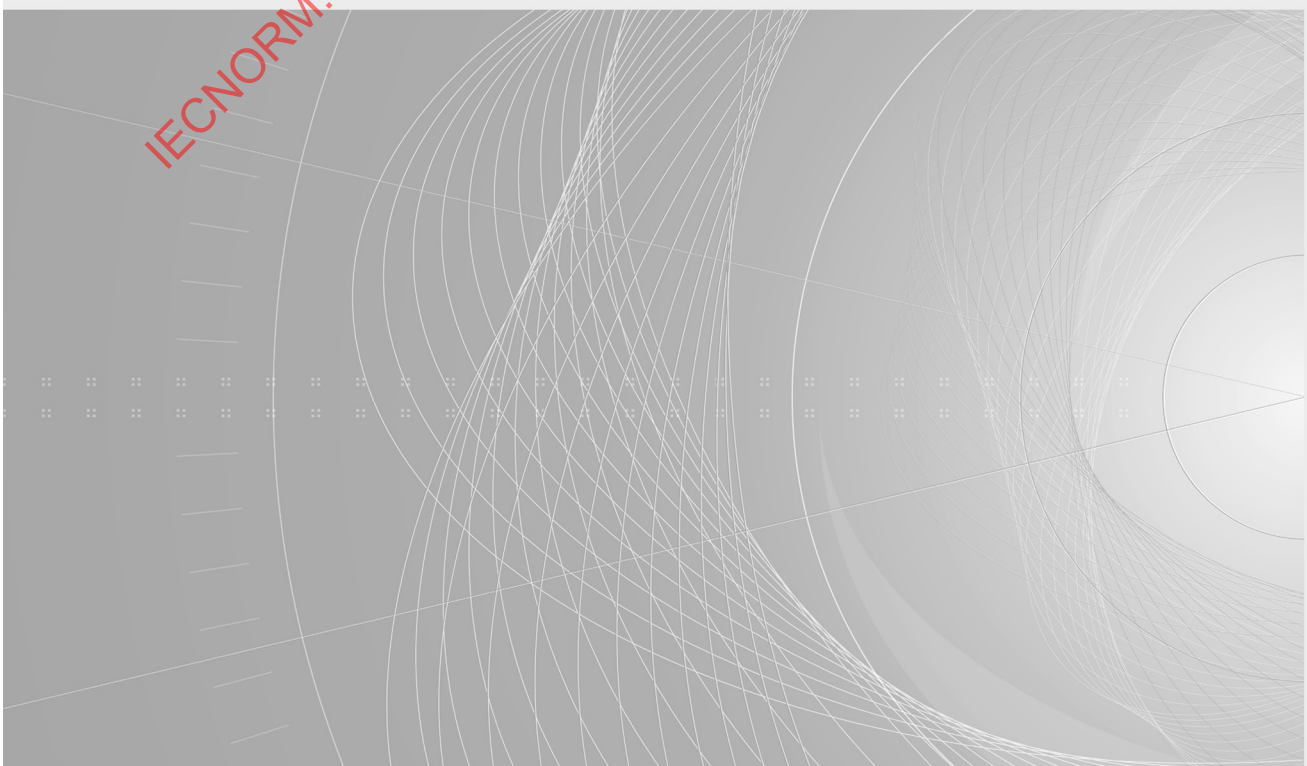
IEC 61158-6-10

Edition 5.0 2023-03

INTERNATIONAL STANDARD

colour inside

Industrial communication networks – Fieldbus specifications – Part 6-10: Application layer protocol specification – Type 10 elements



IEC 61158-6-10:2023-03(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 It also gives information on projects, replaced and withdrawn publications.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 61158-6-10

Edition 5.0 2023-03

INTERNATIONAL STANDARD

colour inside

Industrial communication networks – Fieldbus specifications – Part 6-10: Application layer protocol specification – Type 10 elements

INTERNATIONAL ELECTROTECHNICAL COMMISSION

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-8322-6633-5

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

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-10: Application layer protocol specification – Type 10 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 61158-6-10 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) integration of time-aware system basic functionality;
b) integration of time-aware network functionality;
c) integration of remote service interface functionality;
d) integration of SFP diagnosis functionality;
e) integration of media redundancy ring interconnection basic functionality.

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

Draft	Report on voting
65C/1204/FDIS	65C/1245/RVD

Full information on the voting for the approval of this International Standard 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 the parts of the IEC 61158 series, 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 "http://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 'color inside' logo on the cover page of this publication indicates that it contains colors which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a color printer.

INTRODUCTION

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

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

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

This 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 can 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 a patent. IEC takes no position concerning the evidence, validity, and scope of this patent right.

The holder of these patent rights has assured IEC that s/he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder 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 6-10: Application layer protocol specification – Type 10 elements

1 Scope

1.1 General

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

This part of IEC 61158 provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 10 fieldbus. The term “time-critical” is used to represent the presence of a time window, within which one or more specified actions are required to be completed with a defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This document defines in an abstract way the externally visible behavior provided by the Type 10 fieldbus application layer in terms of:

- the abstract syntax defining the application layer protocol data units conveyed between communicating application entities,
- the transfer syntax defining the application layer protocol data units conveyed between communicating application entities,
- the application context state machine defining the application service behavior visible between communicating application entities, and
- the application relationship state machines defining the communication behavior visible between communicating application entities.

The purpose of this document is to define the protocol provided to:

- define the wire-representation of the service primitives defined in IEC 61158-5-10 and
- define the externally visible behavior associated with their transfer.

This document specifies the protocol of the Type 10 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498-1) and the OSI Application Layer Structure (ISO/IEC 9545).

1.2 Specifications

The principal objective of this document is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-10.

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

1.3 Conformance

This document does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems. Conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

The following documents 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 61131-9, *Programmable controllers – Part 9: Single-drop digital communication interface for small sensors and actuators (SDCI)*

IEC 61158-2:2023, *Industrial communication networks – Fieldbus specifications – Part 2: Physical layer specification and service definition*

IEC 61158-5-10:2023, *Industrial communication networks – Fieldbus specifications – Part 5-10: Application layer service definition – Type 10 elements*

IEC 61158-6-3:2019, *Industrial communication networks – Fieldbus specifications – Part 6-3: Application layer protocol specification – Type 3 elements*

IEC 61158-6-10:2010 [1], *Industrial communication networks – Fieldbus specifications – Part 6-10: Application layer protocol specification – Type 10 elements*

IEC 62439-2:2021, *Industrial communication networks – High availability automation networks – Part 2: Media Redundancy Protocol (MRP)*

IEC TS 60079-47:2021, *Explosive atmospheres – Part 47: Equipment protection by 2-Wire Intrinsically Safe Ethernet concept (2-WISE)*

ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange*

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

ISO/IEC 8822:1994, *Information technology – Open Systems Interconnection – Presentation service definition*

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

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

---

[1] This earlier edition is mentioned here and in the text for legacy purposes.

ISO/IEC 9834-8, *Information technology – Procedures for the operation of object identifier registration authorities – Part 8: Generation of universally unique identifiers (UUIDs) and their use in object identifiers*

ISO/IEC 10646, *Information technology – Universal coded character set (UCS)*

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

ISO/IEC/IEEE 60559:2020, *Information technology – Microprocessor Systems – Floating-Point arithmetic*

ISO 8601-1:2019, *Date and time − Representations for information interchange – Part 1: Basic rules*

IEEE Std 802-2014, *IEEE Standard for Local and metropolitan area networks: Overview and Architecture*

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

IEEE Std 802.1AC-2016, *IEEE Standard for Local and metropolitan area networks – Media Access Control (MAC) Service Definition*

IEEE Std 802.1AS-2020, *IEEE Standard for Local and metropolitan area networks – Timing and Synchronization for Time-Sensitive Applications*

IEEE Std 802.1CB-2017, *IEEE Standard for Local and metropolitan area networks – Frame Replication and Elimination for Reliability*

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*

IEEE Std 802.11-2020, *IEEE Standard for Information Technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*

IEEE Std 802.15.1-2005, *IEEE Standard for Information technology – Local and metropolitan area networks – Specific requirements – Part 15.1a: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Wireless Personal Area Networks (WPAN)*

IETF RFC 768, J. Postel, "User Datagram Protocol", August 1980, available at https://www.rfc-editor.org/info/rfc768 [viewed 2022-10-06]

IETF RFC 791, J. Postel, "Internet Protocol", September 1981, available at https://www.rfc-editor.org/info/rfc791 [viewed 2022-10-06]

IETF RFC 792, J. Postel, "Internet Control Message Protocol", September 1981, available at https://www.rfc-editor.org/info/rfc792 [viewed 2022-10-06]

IETF RFC 826, D. Plummer, "An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", November 1982, available at https://www.rfc-editor.org/info/rfc826 [viewed 2022-10-06]

IETF RFC 1034, P.V. Mockapetris, "Domain names – concepts and facilities", November 1987, available at https://www.rfc-editor.org/info/rfc1034 [viewed 2022-10-06]

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 2022-10-06]

IETF RFC 2131, R. Droms, "Dynamic Host Configuration Protocol", March 1997, available at https://www.rfc-editor.org/info/rfc2131 [viewed 2022-10-06]

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 2022-10-06]

IETF RFC 2236, W. Fenner, "Internet Group Management Protocol, Version 2", November 1997, available at https://www.rfc-editor.org/info/rfc2236 [viewed 2022-10-06]

IETF RFC 2365, D. Meyer, "Administratively Scoped IP Multicast", July 1998, available at https://www.rfc-editor.org/info/rfc2365 [viewed 2022-10-06]

IETF RFC 2474, K. Nichols, S. Blake, F. Baker, D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", December 1998, available at https://www.rfc-editor.org/info/rfc2474 [viewed 2022-10-06]

IETF RFC 2475, S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Services"; December 1998, available at https://www.rfc-editor.org/info/rfc2475 [viewed 2022-10-06]

IETF RFC 2674, E. Bell, A. Smith, P. Langille, A. Rijhsinghani, K. McCloghrie, "Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions", August 1999, available at https://www.rfc-editor.org/info/rfc2674 [viewed 2022-10-06]

IETF RFC 2863, K. McCloghrie, F. Kastenholz, "The Interfaces Group MIB", June 2000, available at https://www.rfc-editor.org/info/rfc2863 [viewed 2022-10-06]

IETF RFC 3418, R. Presuhn, Ed., "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", December 2002, available at https://www.rfc-editor.org/info/rfc3418 [viewed 2022-10-06]

IETF RFC 3535, Schoenwaelder, J., *Overview of the 2002 IAB Network Management Workshop*, May 2003, https://www.rfc-editor.org/info/rfc3535 [viewed 2022-10-21]

IETF RFC 3621, A. Berger, D. Romascanu, "Power Ethernet MIB", December 2003, available at https://www.rfc-editor.org/info/rfc3621 [viewed 2022-10-06]

IETF RFC 4361, T. Lemon, B. Sommerfeld, "Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)", February 2006, available at https://www.rfc-editor.org/info/rfc4361 [viewed 2022-10-06]

IETF RFC 4363, D. Levi, D. Harrington, "Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering, and Virtual LAN Extensions", January 2006, available at https://www.rfc-editor.org/info/rfc4363 [viewed 2022-10-06]

IETF RFC 4604, H. Holbrook, B. Cain, B. Haberman, "Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast", August 2006, available at https://www.rfc-editor.org/info/rfc4604 [viewed 2022-10-06]

IETF RFC 4632, V. Fuller, T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan", August 2006, available at https://www.rfc-editor.org/info/rfc4632 [viewed 2022-10-06]

IETF RFC 4836, E. Beili, "Definitions of Managed Objects for IEEE Std 802.3 Medium Attachment Units (MAUs)", April 2007, available at https://www.rfc-editor.org/info/rfc4836 [viewed 2022-10-06]

IETF RFC 4949, R. Shirey, " Internet Security Glossary, Version 2", August 2007, available at https://www.rfc-editor.org/info/rfc4949 [viewed 2022-10-06]

IETF RFC 5227, S. Cheshire, "IPv4 Address Conflict Detection", July 2008, available at https://www.rfc-editor.org/info/rfc5227 [viewed 2022-10-06]

IETF RFC 5277, Chisholm, S. and H. Trevino, *NETCONF Event Notifications*, July 2008, https://www.rfc-editor.org/info/rfc5277 [viewed 2022-10-21]

IETF RFC 5539, Badra, M., *NETCONF over Transport Layer Security (TLS)*, May 2009, https://www.rfc-editor.org/info/rfc5539 [viewed 2022-10-21]

IETF RFC 5890, J. Klensin, "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", August 2010, available at https://www.rfc-editor.org/info/rfc5890 [viewed 2022-10-06]

IETF RFC 5905, D. Mills, J. Martin, Ed., J. Burbank, W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", June 2010, available at https://www.rfc-editor.org/info/rfc5905 [viewed 2022-10-06]

IETF RFC 6020, Bjorklund, M., Ed., *YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, October 2010, https://www.rfc-editor.org/info/rfc6020 [viewed 2022-10-21]

IETF RFC 6021, Schoenwaelder, J., Ed., *Common YANG Data Types*, October 2010, https://www.rfc-editor.org/info/rfc6021 [viewed 2022-10-21]

IETF RFC 6087, Bierman, A., *Guidelines for Authors and Reviewers of YANG Data Model Documents*, January 2011, https://www.rfc-editor.org/info/rfc6087 [viewed 2022-10-21]

IETF RFC 6110, Lhotka, L., Ed., *Mapping YANG to Document Schema Definition Languages and Validating NETCONF Content*, February 2011, https://www.rfc-editor.org/info/rfc6110 [viewed 2022-10-21]

IETF RFC 6151, S. Turner, L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", March 2011, available at https://www.rfc-editor.org/info/rfc6151 [viewed 2022-10-06]

IETF RFC 6241, Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., *Network Configuration Protocol (NETCONF)*, June 2011, https://www.rfc-editor.org/info/rfc6241 [viewed 2022-10-21]

IETF RFC 6243, Bierman, A. and B. Lengyel, *With-defaults Capability for NETCONF*, June 2011, https://www.rfc-editor.org/info/rfc6243 [viewed 2022-10-21]

IETF RFC 6244, Shafer, P., *An Architecture for Network Management Using NETCONF and YANG*, June 2011, https://www.rfc-editor.org/info/rfc6244 [viewed 2022-10-21]

IETF RFC 6470, Bierman, A., *Network Configuration Protocol (NETCONF) Base Notifications*, February 2012, https://www.rfc-editor.org/info/rfc6470 [viewed 2022-10-21]

IETF RFC 6536, Bierman, A. and M. Bjorklund, *Network Configuration Protocol (NETCONF) Access Control Model*, March 2012, https://www.rfc-editor.org/info/rfc6536 [viewed 2022-10-21]

IETF RFC 6890, M. Cotton, L. Vegoda, R. Bonica, Ed., B. Haberman, "Special-Purpose IP Address Registries", April 2013, available at https://www.rfc-editor.org/info/rfc6890 [viewed 2022-10-06]

IETF RFC 6918, F. Gont, C. Pignataro, "Formally Deprecating Some ICMPv4 Message Types", April 2013, available at https://www.rfc-editor.org/info/rfc6918 [viewed 2022-10-06]

IETF RFC 8342, Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, *Network Management Datastore Architecture (NMDA)*, March 2018, https://www.rfc-editor.org/info/rfc8342 [viewed 2022-10-21]

ITU-T G.781, *Synchronization layer functions for frequency synchronization based on the physical layer*; available at http://www.itu.int/rec/T-REC-G.781 [viewed 2022-10-06]

The Open Group, Publication C706, *Technical standard DCE1.1: Remote Procedure Call,* available at http://www.opengroup.org/onlinepubs/9629399/toc.htm [viewed 2022-10-06]

Metro Ethernet Forum – MEF 10.4:2018, Subscriber Ethernet Service Attributes, available at https://www.mef.net/resources/mef-10-4-subscriber-ethernet-services-attributes [viewed 2022-10-06]

NIST FIPS PUB 180-4, FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, Secure Hash Standard (SHS), August 2015, available at http://dx.doi.org/10.6028/NIST.FIPS.180-4 [viewed 2022-10-06]

NIST FIPS PUB 186-4, FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, Digital Signature Standard (DSS), July 2013, available at http://dx.doi.org/10.6028/NIST.FIPS.186-4 [viewed 2022-10-06]

3 Terms, definitions, abbreviated terms, symbols, and conventions

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

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

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

3.1 Referenced terms and definitions

3.1.1 ISO/IEC 7498-1 terms

For the purposes of this document, the following terms as defined in ISO/IEC 7498-1 apply:

a) application entity

b) application process

c) application protocol data unit

d) application service element

e) application entity invocation
f) application process invocation
g) application transaction
h) real open system
i) transfer syntax

3.1.2 ISO/IEC 8822 terms

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

a) abstract syntax
b) presentation context

3.1.3 ISO/IEC 8824-1 terms

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

a) object identifier
b) type

3.1.4 ISO/IEC 9545 terms

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

a) application-association
b) application-context
c) application context name
d) application-entity-invocation
e) application-entity-type
f) application-process-invocation
g) application-process-type
h) application-service-element
i) application control service element

3.2 Terms and definitions

3.2.1
alarm
activation of an event that indicates a critical state

3.2.2
alarm ack
acknowledgement of an event that indicates a critical state

3.2.3
alarm data object
object(s) which represent(s) critical states referenced by device/ API/ slot/ subslot/ alarm type

3.2.4
allocate
take a resource from a common area and assign that resource for the exclusive use of a specific entity

3.2.5
application
function or data structure for which data is consumed or produced

3.2.6
application data cycle
user-defined time interval required for data-exchange between applications

3.2.7
application layer interoperability
capability of application entities to perform coordinated and cooperative operations using the services of the FAL

3.2.8
application objects
multiple object classes that manage and provide a runtime exchange of PDUs across the network and within the network device

3.2.9
application process
part of a distributed application on a network, which is located on one device and unambiguously addressed

3.2.10
application process identifier
attribute that identifies an application process used in a device

3.2.11
application process object
component of an application process that is identifiable and accessible through an FAL application relationship

Note 1 to entry: Application process object definitions are composed of a set of values for the attributes of their class (see the definition for Application Process Object Class). Application process object definitions may be accessed remotely using the services of the FAL Object Management ASE. FAL Object Management services can be used to load or update object definitions, to read object definitions and to dynamically create and delete application objects and their corresponding definitions.

3.2.12
application process object class
class of application process objects defined in terms of the set of their network-accessible attributes and services

3.2.13
application relationship
cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation

Note 1 to entry: This relationship is activated either by the exchange of application-PDUs or as a result of pre-configuration activities.

3.2.14
application relationship application service element
application-service-element that provides the exclusive means for establishing and terminating all application relationships

3.2.15
application relationship endpoint
context and behavior of an application relationship as seen and maintained by one of the application processes involved in the application relationship

Note 1 to entry: Each application process involved in the application relationship maintains its own application relationship endpoint.

3.2.16
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.

3.2.17
backup
status of an application relationship which indicates that it is in the backup state

Note 1 to entry: An IO AR needs to be established to enter backup state.

3.2.18
behavior
indication of how an object responds to particular events

3.2.19
channel
representation of a single physical or logical link of an input or output application object of a server to the process in order to support addressing of diagnosis information

Note 1 to entry: The channel typically represents a single connector or clamp as a physical interface of a submodule. This reference is used to identify points of failure within diagnosis PDUs and to assign parameter during parameterization.

3.2.20
channel diagnosis
information concerning a specific element of an input or output application object, provided for maintenance purposes

EXAMPLE 1 Wire break

EXAMPLE 2 Short circuit

3.2.21
class
set of objects, all of which represent the same kind of system component

Note 1 to entry: A class is a generalization of an object; a template for defining variables and methods. All objects in a class are identical in form and behavior, but usually contain different data in their attributes.

3.2.22
class attributes
attribute that is shared by all objects within the same class

3.2.23
class code
unique identifier assigned to each object class

3.2.24
class 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: A class specific object is unique to the object class which defines it.

3.2.25
client
a) object which uses the services of another (server) object to perform a task

b) initiator of a PDU to which a server reacts

3.2.26
common profile
collection of device independent information and functionality providing consistency between all devices

3.2.27
communication data object
object(s) which is(are) parameter(s) of communication relationships and referenced by device/ slot/ subslot/ index

3.2.28
configuration check
comparison of the expected IO Data object structuring of the client with the real IO Data object structuring to the server in the startup phase

3.2.29
configuration fault
unacceptable difference between the expected IO Data object structuring and the real IO Data object structuring, as detected by the server

3.2.30
configuration identifier
representation of a portion of IO Data of a single input- and/or output-module of a server

3.2.31
consume
act of receiving data from a provider

3.2.32
consumer
node or sink receiving data from a provider

3.2.33
context management
network-accessible information (communication objects) that supports managing the operation of the fieldbus system, including the application layer

Note 1 to entry: Managing includes functions such as controlling, monitoring and diagnosing.

3.2.34
conveyance path
unidirectional flow of APDUs in an application relationship

3.2.35
cyclic
repetitive in a regular manner

3.2.36
data consistency
means for coherent transmission and access of the input- or output-data object between and within client and server

3.2.37
device
physical hardware connected to the link

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

3.2.38
device ID
vendor assigned device type identification

3.2.39
device profile
collection of device dependent information and functionality providing consistency between similar devices of the same device type

3.2.40
diagnosis data object
object(s) which contain(s) diagnosis information referenced by device/ API/ slot/ subslot/ index

3.2.41
diagnosis information
all data available at the server for maintenance purposes

3.2.42
dynamic reconfiguration
change of IO Data objects without interruption of an established Application Relation and continuous updating of unchanged IO Data objects

Note 1 to entry: An IO AR of an AR-set needs to be established for reconfiguration.

3.2.43
endpoint
one of the communicating entities involved in a connection

3.2.44
engineering
abstract term that characterizes the client application or device responsible for configuring an automation system

3.2.45
error
discrepancy between a computed, observed, or measured value or condition and the specified or theoretically correct value or condition

3.2.46
error class
general grouping of related error definitions and corresponding error codes

3.2.47
error code
identification of a specific type of error within an error class

3.2.48
event
instance of a change of conditions

3.2.49
extended channel diagnosis
information concerning a specific element of a specific application object, provided for maintenance purposes

EXAMPLE 1 Plastic Optical Fiber, damping level

EXAMPLE 2 Congestion loss, number of frames

3.2.50
frame
unit of data transmission on an IEEE Std 802 Local Area Network (LAN) that conveys a Media Access Control (MAC) Protocol Data Unit (MPDU)

[SOURCE: IEEE Std 802.1Q:2018, 3.95, modified – deleted “A” at the beginning and full stop at the end to align with IEC style]

3.2.51
gating cycle
user-defined time interval derived from Working or Local Clock and used to control access to Ethernet (gate control list)

3.2.52
identification data object
object(s) that contain(s) information about device, module and submodule manufacturer and type referenced by device/ API/ slot/ subslot/ index

3.2.53
implicit AR endpoint
AR endpoint that is defined locally within a device existing by default

3.2.54
inbound
input communication relation from IO device to IO controller

3.2.55
index
address of a record data object within an application process

3.2.56
instance
actual physical occurrence of an object within a class that identifies one of many objects within the same object class

3.2.57
instance attributes
attribute that is unique to an object instance and not shared by the object class

3.2.58
instantiated
object instance that has been created in a device

3.2.59
invocation
act of using a service or other resource of an application process

Note 1 to entry: Each invocation represents a separate thread of control that can be described by its context. Once the service completes, or use of the resource is released, the invocation ceases to exist. For service invocations, a service that has been initiated but not yet completed is referred to as an outstanding service invocation. Also, for service invocations, an Invoke ID may be used to unambiguously identify the service invocation and differentiate it from other outstanding service invocations.

3.2.60
IO controller
controlling device, which acts as client for several IO devices (field devices)

Note 1 to entry: This entity is typically found in a programmable controller or distributed control system.

3.2.61
IO Data object
object designated to be transferred cyclically for the purpose of processing and referenced by device/ API/ slot/ subslot

3.2.62
IO device
field device which acts as server for IO operation

Note 1 to entry: This entity is typically found in a sensor and/or actuator.

3.2.63
IO parameter server
server for application parameter of IO devices (client)

Note 1 to entry: This entity is typically found in a device used to backup parameter data and to log online changes of device parameter.

3.2.64
IO supervisor
engineering device which manages commissioning and diagnosis of an IO system

Note 1 to entry: This entity is typically found in an engineering tool.

3.2.65
IO system
system composed of IO controllers and all associated IO devices

3.2.66
Isochronous mode
applications of IO controllers and IO devices operating tightly synchronized with a jitter of less than 1 µs

3.2.67
member
piece of an attribute that is structured as an element of an array

3.2.68
message
ordered series of octets intended to convey information

Note 1 to entry: Used to convey information between peers at the application layer.

Note 2 to entry: Messages are built upon packets, which are built upon frames.

[SOURCE: derived from ISO/IEC 2382:2015]

3.2.69
method
synonym for an operational service which is provided by the server ASE and invoked by a client

3.2.70
module
hardware or logical component of a physical device

3.2.71
network
set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways

3.2.72
NME domain
set of nodes where every node contains the same information about the responsible configuration entity

3.2.73
object
abstract representation of a particular component within a device, usually a collection of related data (in the form of variables) and methods (procedures) for operating on that data that has a clearly defined interface and behavior

3.2.74
object specific service
service unique to the object class which defines it

3.2.75
operate
status of the IO controller which indicates that the control algorithm is currently running

3.2.76
outbound
output communication relation from an IO controller to an IO device

3.2.77
packet
logical grouping of information used to describe a unit of data at any layer to convey the upper layer user data to its peer layer

Note 1 to entry: A packet is identical to the PDU at each layer in terms of the OSI reference model. A data-link layer packet is a frame. Messages are built upon packets, which are built upon frames.

3.2.78
peer
role of an AR endpoint in which it is capable of acting as both client and server

3.2.79
physical device
automation or other network device

3.2.80
point-to-point connection
connection that exists between exactly two application objects

3.2.81
primary
status of the IO AR that indicates that it is in the operating state

Note 1 to entry: Besides a primary IO AR a backup IO AR may exist. For example used for redundancy and dynamic reconfiguration of IO Data.

3.2.82
provider
node or source sending data to one or many consumers

3.2.83
PTCP subdomain
system composed of IO controller(s) and all its associated IO devices synchronized by the associated PTCP master

3.2.84
qualified channel diagnosis
information concerning a specific element of a specific application object, provided for maintenance purposes with a parameterized severity

EXAMPLE 1 Severity maintenance required

EXAMPLE 2 Severity fault

3.2.85
record data object
object(s) which are already pre-processed and transferred acyclically for the purpose of information or further processing and referenced by device/ API/ slot/ subslot/ index

3.2.86
RedApplication
application supporting redundant or high availability operation

3.2.87
resource
processing or information capability

3.2.88
run
status of the IO controller which indicates that the control algorithm is currently operating

3.2.89
server
a) role of an AREP in which it returns a confirmed service response APDU to the client that initiated the request

b) object which provides services to another (client) object

3.2.90
service
operation or function that an object and/or object class performs upon request from another object and/or object class

3.2.91
slot
address of a structural unit within an IO device

Note 1 to entry: Within a modular device, a slot typically addresses a physical module. Within compact devices, a slot typically addresses a logical function or virtual module.

3.2.92
stop
status of the application which indicates that the control algorithm is currently not running

3.2.93
submodule
hardware or logical component of a module

3.2.94
subslot
address of a structural unit within a slot

Note 1 to entry: A subslot can address a physical interface for submodules within a module. Generally, a subslot is a second level to structure data within a device.

3.2.95
time-aware system
end station or bridge synchronized using IEEE Std 802.1AS controlling its access to the network

Note 1 to entry: Time-aware systems are devices which support multiple VLANs, multiple traffic classes, stream categories, time-controlled network access, ingress rate limiting and traffic engineering.

Note 2 to entry: Non-time-aware systems are devices which can support multiple VLANs, multiple traffic classes, stream categories, time-controlled network access, ingress rate limiting and traffic engineering.

3.2.96
vendor ID
central administrative number used as manufacturer identification

3.3 Abbreviated terms and symbols

3.3.1 Abbreviated terms and symbols for services

Cnf	Confirmation
Ind	Indication
Req	Request
Rsp	Response

3.3.2 Abbreviated terms and symbols for distributed I/O

ACD	Access Control Decision
AE	Application Entity
AL	Application Layer
ALME	Application Layer Management Entity
ALP	Application Layer Protocol
ALPMI	Alarm Protocol Machine Initiator
ALPMR	Alarm Protocol Machine Responder
AP	Application Process
APDU	Application Protocol Data Unit
API	Application Process Identifier
APM	Acyclic Protocol Machine
APMR	Acyclic Protocol Machine Receiver
APMS	Acyclic Protocol Machine Sender
APO	Application Object
AR	Application Relationship
AREP	Application Relationship End Point
ARP	Address Resolution Protocol
ARPM	Address Resolution Protocol Machine
ASCII	American Standard Code for Information Interchange
ASDU	Application Service Data Unit
ASE	Application Service Element
BLOB	Binary Large OBject
BMA	Best-Master-Algorithm Protocol Machine
BMC	Best Master Clock
BNME	Best Network Management Entity
CIM	Communication Interface Management
CLRPC	Connectionless Remote Procedure Call

CL-RPC	Connectionless Remote Procedure Call
CM	Context Management
CMCTL	Context Management Protocol Machine Controller
CMDEV	Context Management Protocol Machine Device
CMDEV_DA	Context Management Device Access Protocol Machine Device
CMDMC	Context Management Discovery Multicast Communication Protocol Machine
CMINA	Context Management IP and Name Assignment Protocol Machine
CMIO	Context Management Input Output Protocol Machine Device
CMPBE	Context Management Prm Begin End Protocol Machine Device
CMRDR	Context Management Read Record Responder Protocol Machine Device
CMRPC	Context Management RPC Device Protocol Machine
CMRS	Context Management Reporting System Protocol Machine Device
CMRSI	Context Management RSI Device Protocol Machine
CMSM	Context Management Surveillance Protocol Machine Device
CMSRL	Context Management System Redundancy Layer Device
CMSRL_AL	Context Management System Redundancy Layer Alarm Device Protocol Machine
CMSU	Context Management Startup Protocol Machine Device
CMWRR	Context Management Write Record Responder Protocol Machine Device
CPM	Consumer protocol machine for cyclic transmitted data
CR	Communication Relationship
CREP	Communication Relationship End Point
CT	Cut Through switching
CTLDINA	Context Management Discovery, IP and Name Assignment Protocol Machine
CTLIO	Context Management Input Output Protocol Machine Controller
CTLPBE	Context Management Prm Begin End Protocol Machine Controller
CTLRDI	Context Management Read Record Initiator Protocol Machine Controller
CTLRDR	Context Management Read Record Responder Protocol Machine Controller
CTLRPC	Context Management RPC Protocol Machine Controller
CTLRSI	Context Management RSI Protocol Machine Controller
CTLSM	Context Management Surveillance Protocol Machine Controller
CTLSRL	Context Management System Redundancy Layer Controller
CTLSU	Context Management Startup Protocol Machine Controller
CTLWRI	Context Management Write Record Initiator Protocol Machine Controller
CTLWRR	Context Management Write Record Responder Protocol Machine Controller
DCE	OSF Distributed Computing Environment
DCP	Discovery and basic Configuration Protocol
DCPHMCS	DCP Hello Multicast Sender Protocol Machine
DCPHMRC	DCP Hello Multicast Receiver Protocol Machine
DCPMCR	DCP Multicast Receiver Protocol Machine
DCPMCS	DCP Multicast Sender Protocol Machine
DCPUCR	DCP Unicast Receiver Protocol Machine
DCPUCS	DCP Unicast Sender Protocol Machine
DEFRAG	Defragmentation Protocol Machine

DELAY_REQ	Line Delay Request Protocol Machine
DELAY_RSP	Line Delay Response Protocol Machine
DEMUX	Network Access Receiver
DFP	Dynamic Frame Packing
DFP_RELAY	DFP Protocol Machine
DFP_RELAY_IN_STORAGE	DFP Inbound Storage Protocol Machine
DFP_RELAY_INBOUND	DFP Inbound Protocol Machine
DFP_RELAY_OUTBOUND	DFP Outbound Protocol Machine
DHCP	Dynamic Host Configuration Protocol
DIM	Device Interface Module
DLC	Data Link Connection
DLL	Data Link Layer
DLPDU	Data Link Protocol Data Unit
DLSDU	Data Link Service Data Unit
DMPM	DLL Mapping Protocol Machine
DNS	Domain Name Service
DTE	Data Terminal Equipment
FAL	Fieldbus Application Layer
FDB	Filtering Data Base
FIFO	First In First Out
FQDN	Fully Qualified Domain Name
FRAG	Fragmentation Protocol Machine
FSPM	FAL Service Protocol Machines
FSPMCTL	FSPM Controller
FSPMDEV	FSPM Device
GBSM	Get Best Sync Master Protocol Machine
GSDML	General Station Description Markup Language
I&M	Identification and Maintenance Profile
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ID	Identifier
IEC	International Electrotechnical Commission
IFW	RT_CLASS_3 Forwarding Protocol Machine
IOC	Input Output Controller
IOCR	IO Communication Relation
IOCS	Input Output Object Consumer Status
IOD	Input Output Device
IOMCR	IO Multicast Communication Relation
IOPS	Input Output Object Provider Status
IOS	Input Output Supervisor
IP	Internet Protocol
IR	Isochronous Relay
IRT	Isochronous Real Time Protocol
ISO	International Organization for Standardization
IsoM	Isochronous Mode
LED	Light Emitting Diode

LLDP	Link Layer Discovery Protocol
LME	Layer Management Entity
LMPM	Data Link Layer Mapping Protocol Machine
lsb	Least Significant Bit
MAC_RELAY	Forwarding Protocol Machine
MCPM	Multicast Consumer Protocol Machine for cyclic transmitted data
MCR	Multicast Communication Relation
MPPM	Provider Protocol Machine for cyclic transmitted data (PPM) using Multicast communication relation
msb	most significant bit
MUX	Network Access Sender Protocol Machine
NAP	Network Access Point
NCA	Network Computing Architecture
NCE	Network Configuration Engine
NME	Network Management Entity
NUE	Network Update Engine
OID	Object IDentifier
OSF	Open Software Foundation
OSI	Open Systems Interconnect
PCE	Path Compute Engine
PDU	Protocol Data Unit
PL	Physical Layer
PPM	Provider Protocol Machine for cyclic transmitted data
PTCP	Precision Transparent Clock Protocol
QPSM	IEEE Std 802.1Q mechanism-based Port State Machine
QoS	Quality of Service
RED_RELAY	Realtime Class 3 Forwarding Protocol Machine
RNG	Random Number Generator
RPC	Remote Procedure Call
RS	Reporting System
RSI	Remote Service Interface
RSM	Best Remote Sync Master Protocol Machine
RSO	Reporting System Observer
RSTP	Rapid Spanning Tree Protocol
RT	Real Time (protocol)
RT_CLASS_UDP	Real Time Class UDP (supports Layer 3 transport)
RT_CLASS_1	Real Time Class 1 (Layer 2 transport)
RT_CLASS_2	Real Time Class 2 (Layer 2 transport)
RT_CLASS_3	Real Time Class 3 (Layer 2 transport)
RTA	Real Time Acyclic (protocol)
RTC	Real Time Cyclic (protocol)
RTC3PSM	Real Time Class 3 Port State Machine
SCHEDULER	Scheduler Protocol Machine
SDU	Service Data Unit
SFP	Small Form-factor Pluggable
SYN_BMA	Best-Master-Algorithm Protocol Machine
SYN_BMA_GBSM	Get Best Sync Master

SYN_BMA_RSM	Best Remote Sync Master
SYN_MPSM	PTCP Master Protocol Machine
SYN_SPSM	PTCP Slave Protocol Machine
SYNC_RELAY	Sync Relay Protocol Machine
TAS	Time Aware Shaper
TDE	Topology Discovery Engine
TLV	Type Length Value (coding rule)
TSN	Time Sensitive Networking
UDP	User Datagram Protocol
USI	User Structure Identifier
UUID	Universal Unique Identifier

3.3.3 Abbreviated terms and symbols for IEC 62439-2

MIC	Media Redundancy Interconnection Client
MIM	Media Redundancy Interconnection Manager
MRA	Media Redundancy Automanager
MRC	Media Redundancy Client
MRM	Media Redundancy Manager
MRP	Media Redundancy Protocol
MRPD	Media Redundancy with Planned Duplication of frames

NOTE The Media Redundancy with Planned Duplication of frames is done by the sender using different paths for the adjunctive frames.

3.3.4 Abbreviated terms and symbols for IEC/IEEE 60802

CNC	Centralized Network Configuration
CUC	Centralized User Configuration

3.3.5 Abbreviated terms and symbols for IEEE Std 802.1CB

RTI	Redundancy Tag Information

3.3.6 Abbreviated terms and symbols for IEEE Std 802.1Q

CBS	Committed burst size
CBSA	Credit Based Shaping Algorithm
CIR	Committed information rate
CIST	Common and Internal Spanning Tree
C-VLAN	Customer – Virtual Local Area Network
DEI	Drop Eligible Indicator
ETS	Enhanced Transmission Selection
IST	Internal Spanning Tree
IVL	Independent VLAN Learning
MSTID	Multiple Spanning Tree Instance Identifier
PCP	Priority Code Point
QBTSA	Queue Based Transmission Selection Algorithm
SVL	Shared VLAN Learning

TCI	Tag Control Information
TE-MST	Traffic engineered – Multiple Spanning Tree
VID	VLAN Identifier
VLAN	Virtual Local Area Network

3.3.7 Abbreviated terms and symbols for IEEE Std 802.3

DA	Destination address
eMAC	Express MAC
FCS	Frame check sequence
IPG	Inter Packet Gap
LLC	Logical Link Control
LT	Length/Type
MAC	Media Access Control
MAU	Medium Attachment Unit
MDI	Media Dependent Interface
MII	Media Independent Interface
pMAC	Preemptable MAC
RGMII	Reduced Gigabit Media Independent Interface
SA	Source address
S&F	Store and Forward

3.3.8 Abbreviated terms and symbols for IETF RFC 2474

DSCP	Differentiated Services Code Point
ECN	Explicit Congestion Notification

3.3.9 Abbreviated terms and symbols for IETF RFC 4291

EUI-64	64-Bit Extended Unique Identifier

3.4 Conventions

3.4.1 General concept

The FAL is defined as a set of object-oriented ASEs. Each ASE is specified in a separate clause. Each ASE specification is composed of three parts: its class definitions, its services, and its protocol specification. The first two parts are contained in IEC 61158-5-10. The protocol specification for each of the ASEs is defined in this document.

The class definitions define the attributes of the classes supported by each ASE. The attributes are accessible from instances of the class using the Management ASE services specified in IEC 61158-5-10. The service specification defines the services that are provided by the ASE.

This document uses the descriptive conventions given in ISO/IEC 10731.

3.4.2 Conventions for distributed I/O

3.4.2.1 Abstract syntax conventions

PDUs are described as octets or groups of octets:

a) Groups of octets separated by a comma appear in the order they are transferred. If optional octets are not present, the following octets appear without a gap.

b) If octets or groups of octets are grouped within “{ }”, the order is arbitrary.

c) If octets or groups of octets are marked with “*”, they may appear more than once. If the asterisk is used within a “{ }” section, octets may appear mixed with other octets or group of octets of this section.

d) Octets can be grouped or values can be assigned within “( )”.

e) If octets or groups of octets are grouped within “[ ]”, the group can be omitted. This doesn’t imply that the support of these octets is optional.

f) Complex APDUs may be built by means of substitutions (sub-structures).

g) Exclusive selections of octets or groups of octets are separated by “^”

NOTE 1 Formal PDU example

APDU = Octet1, OctetGroup1, [Octet2], [Octet3], {[OctetGroup2*], OctetGroup3 ^ Octet4}

According to this the following variants are valid on the wire (non-exhaustive):

Variant 1: Octet1, OctetGroup1, Octet2, Octet3, OctetGroup2, OctetGroup3

Variant 2: Octet1, OctetGroup1, Octet2, Octet3, OctetGroup2, OctetGroup2, OctetGroup2, OctetGroup3

Variant 3: Octet1, OctetGroup1, OctetGroup2, OctetGroup2, OctetGroup2, OctetGroup3, OctetGroup2

Variant 4: Octet1, OctetGroup1, OctetGroup2, OctetGroup3, OctetGroup2, OctetGroup2, OctetGroup2, OctetGroup2

Variant 5: Octet1, OctetGroup1, Octet3, Octet4

NOTE 2 The arbitrary order implies that groups of octets are characterized by a special header that is described within the coding rules.

NOTE 3 The APDU syntax for RTA- and RTC-PDU implies that according to the maximum DLSDU an APDU does not exceed 1 440 octets in total.

NOTE 4 The APDU syntax for CL-RPC implies that an IO controller supports a minimal ASDU size of 4 096 octets in total and does not exceed 2^{32}-64 octets in total. The minimal ASDU size is derived from the expected size of configuration, parameter, and diagnosis data of an enhanced IO device.

3.4.2.2 Conventions for the encoding of reserved bits and octets

The term “reserved” is used to describe bits in octets or whole octets. All bits or octets that are reserved should be set to zero at the sending side and shall not be checked at the receiving side except it is explicitly stated or if the reserved bits or octets are checked by a state machine.

The term “reserved” is also used to indicate that certain values within the range of a parameter are reserved for future extensions. In this case the reserved values should not be used at the sending side and shall not be checked at the receiving side except it is explicitly stated or if the reserved values are checked by a state machine.

3.4.2.3 Conventions for the common coding of specific field octets

3.4.2.3.1 General

APDUs may contain specific fields that carry information in a primitive and condensed way. These fields shall be coded in the order according to Figure 1, Figure 2, Figure 3, Figure 4, Figure 5, Figure 6, Figure 7, Figure 8, Figure 9, Figure 10, Figure 11, Figure 12, Figure 13, Figure 14, Figure 15, and Figure 16.

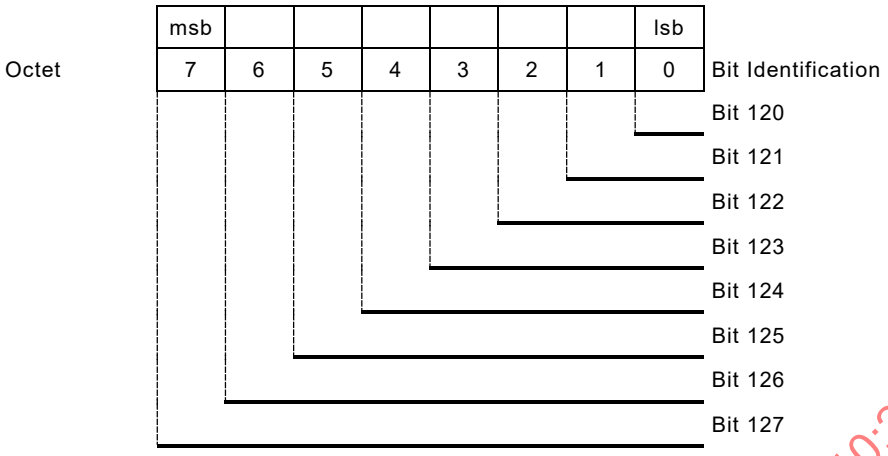


Figure 1 – Common structure of specific fields for octet 1

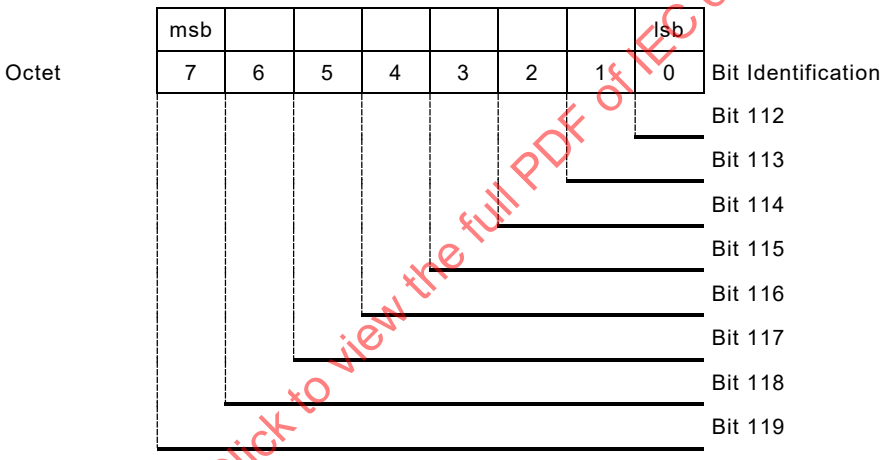


Figure 2 – Common structure of specific fields for octet 2

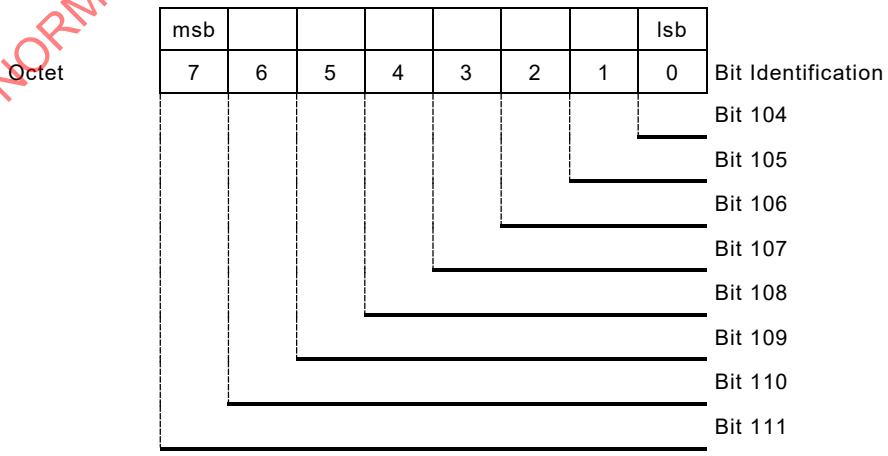


Figure 3 – Common structure of specific fields for octet 3

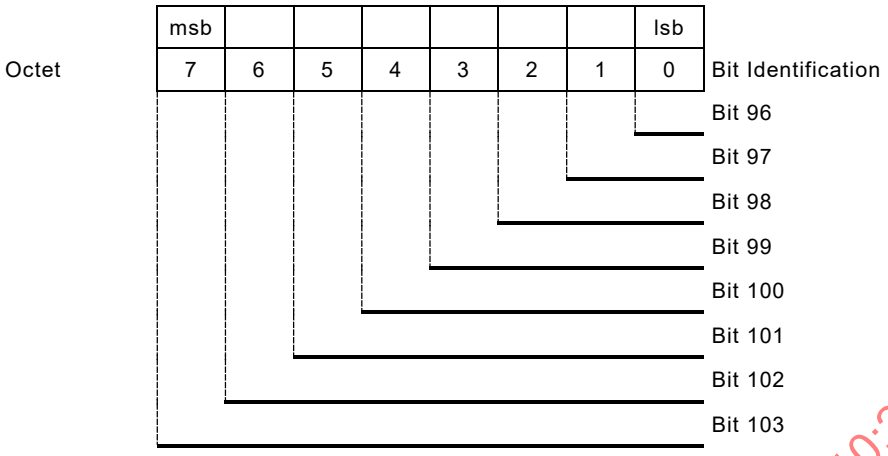


Figure 4 – Common structure of specific fields for octet 4

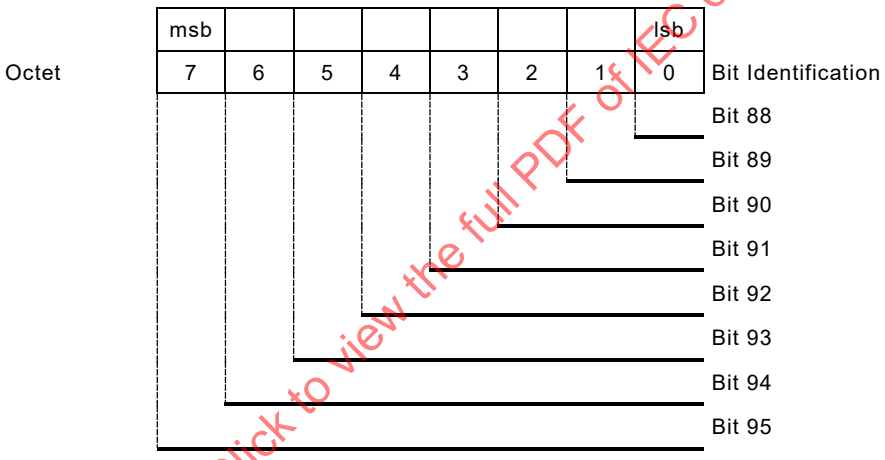


Figure 5 – Common structure of specific fields for octet 5

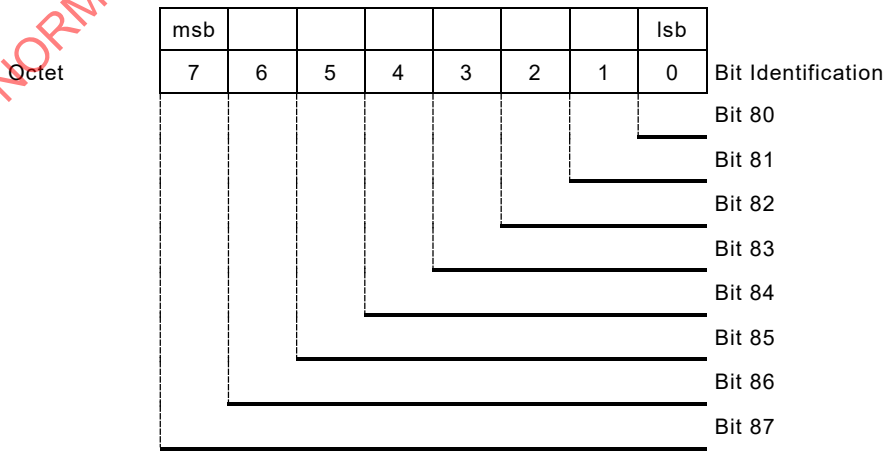


Figure 6 – Common structure of specific fields for octet 6

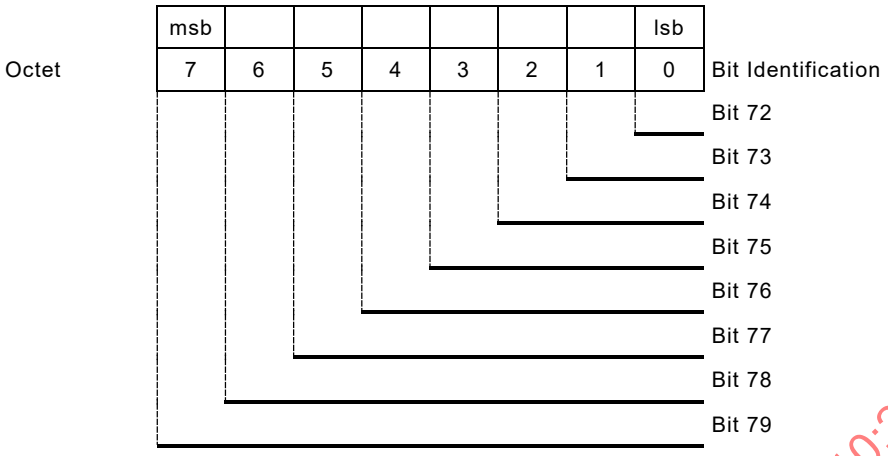


Figure 7 – Common structure of specific fields for octet 7

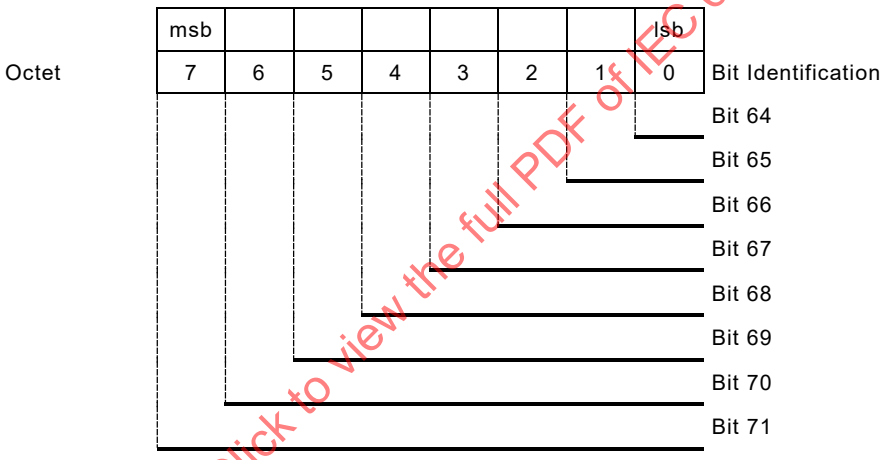


Figure 8 – Common structure of specific fields for octet 8

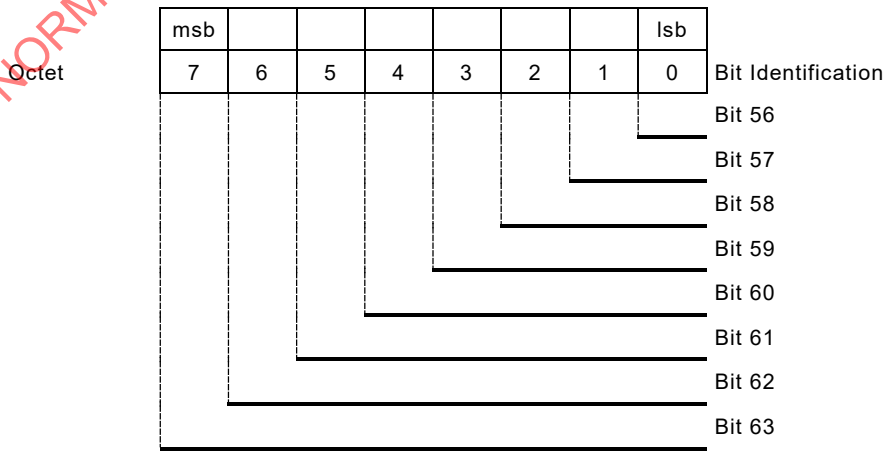


Figure 9 – Common structure of specific fields for octet 9

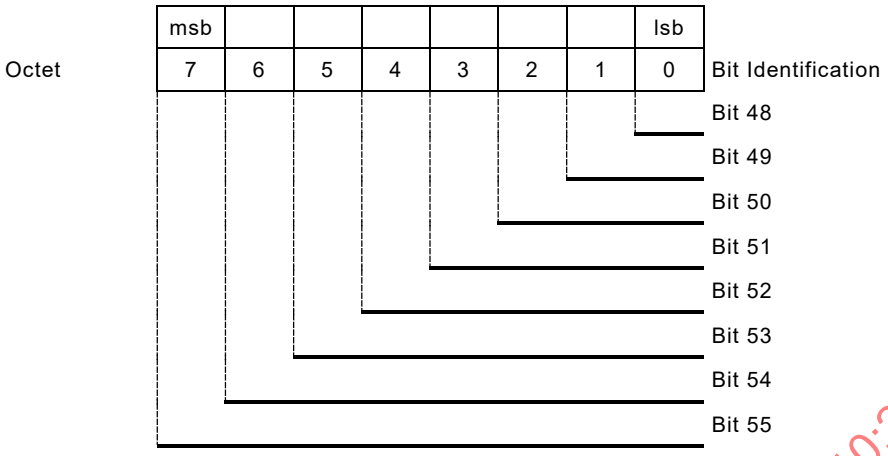


Figure 10 – Common structure of specific fields for octet 10

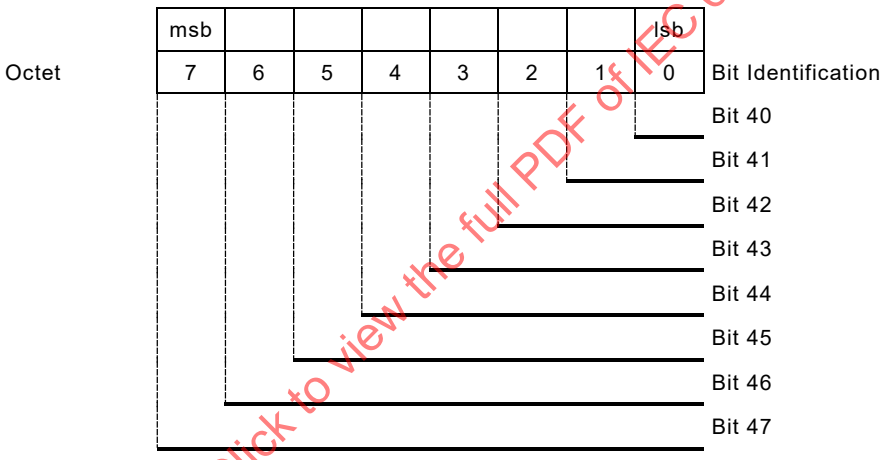


Figure 11 – Common structure of specific fields for octet 11

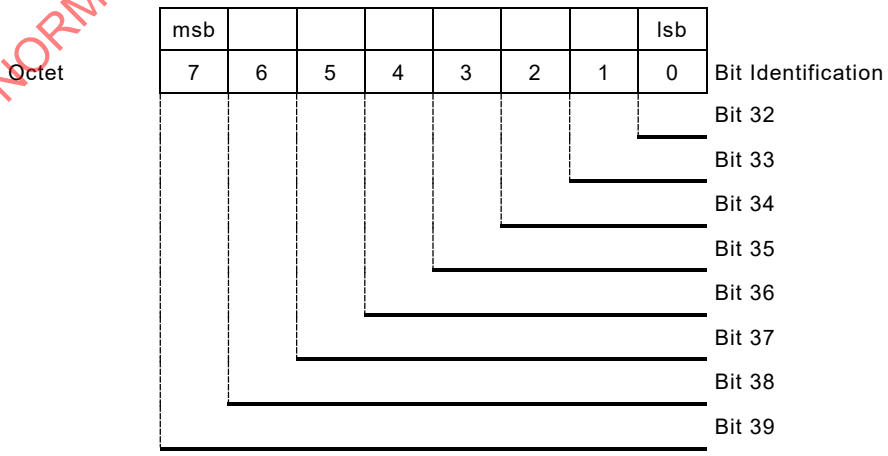


Figure 12 – Common structure of specific fields for octet 12

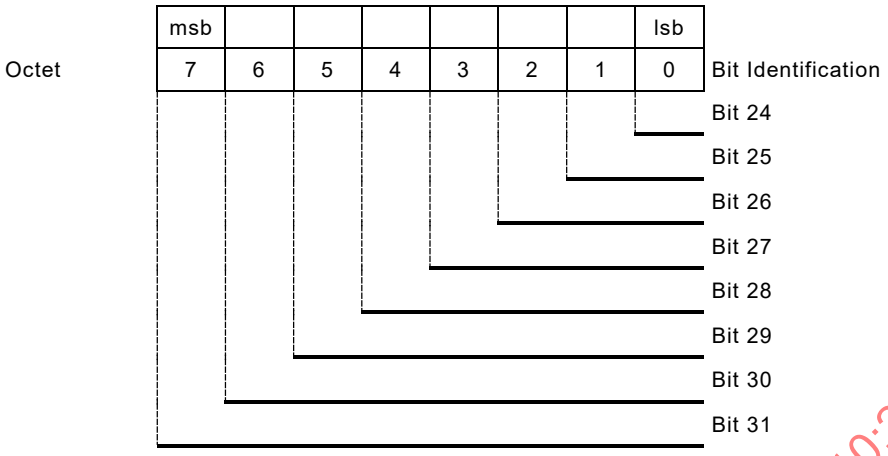


Figure 13 – Common structure of specific fields for octet 13

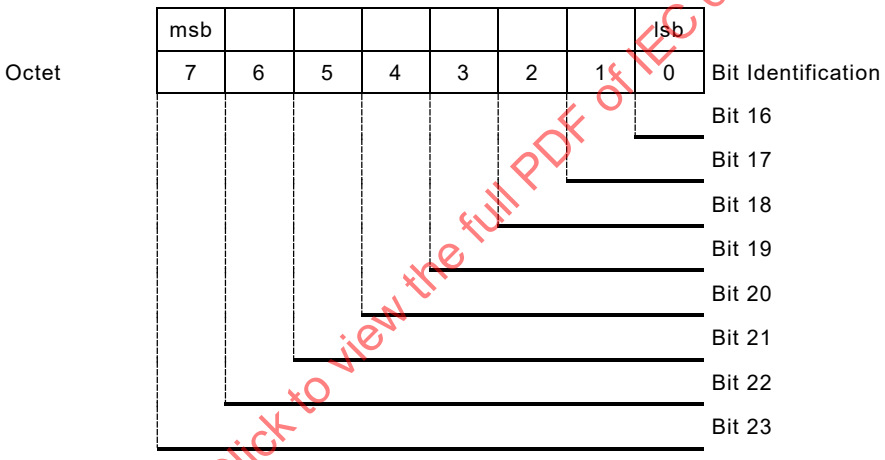


Figure 14 – Common structure of specific fields for octet 14

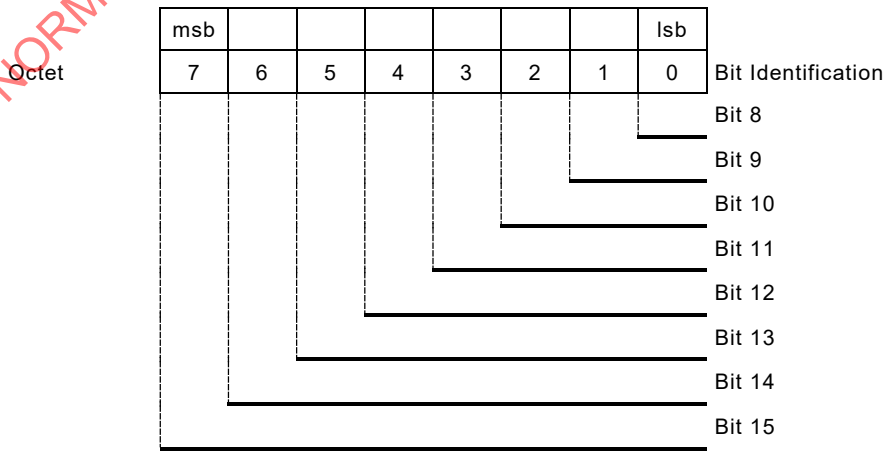


Figure 15 – Common structure of specific fields for octet 15

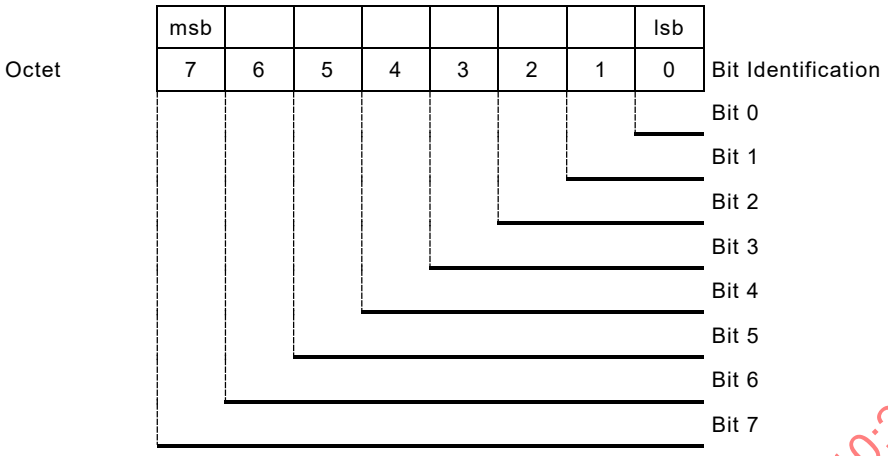


Figure 16 – Common structure of specific fields for octet 16

3.4.2.3.2 Rules for the handling of bits and groups of bits

Bits may be grouped as groups of bits. Each bit or group of bits shall be addressed by its Bit Identification (for example Bit 0, Bit 1 to Bit 4, Bit 120 to Bit 127). The position within the octet shall be according to Figure 1 to Figure 16. Alias names may be used for each bit or group of bits or they may be marked as reserved. The grouping of individual bits shall be in ascending order without gaps.

The values for a group of bits may be represented as binary, decimal or hexadecimal values. This value shall only be valid for the grouped bits and can only represent the whole octet if all 128 bits are grouped. Decimal or hexadecimal values shall be transferred in binary values so that the bit with the highest number of the group represents the most significant bit (msb) concerning the grouped bits.

3.4.2.3.3 Conventions for the common codings of specific fields consisting of one octet

APDUs may contain specific fields that carry information in a primitive and condensed way.

These fields shall be coded in the order according to Table 1.

Rules from 3.4.2.3.2 apply.

Table 1 – One octet

Element	Meaning
octet 1	See Figure 16

EXAMPLE 1 Description and relation for the specific field octet:

Bit 0: reserved.

Bit 1 – Bit 3: Reason_Code. The decimal value 2 for the Reason_Code means general error.

Bit 4 – Bit 7: shall be set to one.

The octet that is constructed according to the description above looks as follows:

(msb) Bit 7 = 1,

Bit 6 = 1,

Bit 5 = 1,

Bit 4 = 1,

Bit 3 = 0,

Bit 2 = 1,

Bit 1 = 0,

(lsb) Bit 0 = 0.

The bit combination “0-1-0” for Bit 1 – Bit 3 equals the decimal value 2.

3.4.2.3.4 Conventions for the common codings of specific fields consisting of two subsequent octets

APDUs may contain specific fields that carry information in a primitive and condensed way.

These fields shall be coded in the order according to Table 2.

Rules from 3.4.2.3.2 apply.

Table 2 – Two subsequent octets

Element	Meaning
octet 1 (high)	See Figure 15
octet 2 (low)	See Figure 16

3.4.2.3.5 Conventions for the common coding of specific fields consisting of four subsequent octets

APDUs may contain specific fields that carry information in a primitive and condensed way.

These fields shall be coded in the order according to Table 3.

Rules from 3.4.2.3.2 apply.

Table 3 – Four subsequent octets

Element	Meaning
octet 1 (high)	See Figure 13
octet 2	See Figure 14
octet 3	See Figure 15
octet 4 (low)	See Figure 16

3.4.2.3.6 Conventions for the common coding of specific fields consisting of eight subsequent octets

APDUs may contain specific fields that carry information in a primitive and condensed way.

These fields shall be coded in the order according to Table 4.

Rules from 3.4.2.3.2 apply.

Table 4 – Eight subsequent octets

Element	Meaning
octet 1 (high)	See Figure 9
octet 2	See Figure 10
octet 3	See Figure 11
octet 4	See Figure 12
octet 5	See Figure 13
octet 6	See Figure 14
octet 7	See Figure 15
octet 8 (low)	See Figure 16

3.4.2.3.7 Conventions for the common coding of specific fields consisting of sixteen subsequent octets

APDUs may contain specific fields that carry information in a primitive and condensed way.

These fields shall be coded in the order according to Table 5.

Rules from 3.4.2.3.2 apply.

Table 5 – Sixteen subsequent octets

Element	Meaning
octet 1 (high)	See Figure 1
octet 2	See Figure 2
octet 3	See Figure 3
octet 4	See Figure 4
octet 5	See Figure 5
octet 6	See Figure 6
octet 7	See Figure 7
octet 8	See Figure 8
octet 9	See Figure 9
octet 10	See Figure 10
octet 11	See Figure 11
octet 12	See Figure 12
octet 13	See Figure 13
octet 14	See Figure 14
octet 15	See Figure 15
octet 16 (low)	See Figure 16

3.4.3 Conventions used in state machines

The protocol sequences are described by means of State Machines.

In state diagrams states are represented as boxes and state transitions are shown as arrows. Names of states and transitions of the state diagram correspond to the names in the textual listing of the state transitions.

The initialization of all parameters (for example default values) of a state machine will be realized automatically during the instantiation of the machine (new instance). Implicit transition from POWER-ON to first state is done by calling the INIT service. If it is necessary to have different instances of one machine type, different parameters will be set by the INIT service.

The layout of a Machine description is shown in Table 6.

The textual listing of the state transitions is structured as follows:

– The first column contains the name of the transition.
– The second column defines the current state.
– The third column contains an optional event followed by conditions starting with a "/" as first line character and finally followed by the actions starting with a "=>" as first line character.
– The last column contains the next state.

If the event occurs and the conditions are fulfilled the transition fires, for example the actions are executed, and the next state is entered.

The meaning of the elements of a State Machine Description is shown in Table 7.

Table 6 – State machine description elements

#	Current state	Event /Condition => Action	Next state

Table 7 – Description of state machine elements

Description element	Meaning
#	Name or number of the state transition.
Current state Next state	Name of the given state.
Event	Name or description of the event.
/Condition	Boolean expression. The preceding "/" is not part of the condition.
=> Action	List of assignments and service or function invocations. The preceding "=>" is not part of the action.

The conventions used in the state machines are shown in Table 8.

Table 8 – Conventions used in state machines

Convention	Meaning
:=	Value of an item on the left is replaced by value of an item on the right. If an item on the right is a parameter, it derives from the primitive shown as an input event.
xxx	A parameter name. Example: Identifier := reason meaning value of a ‘reason’ parameter is assigned to a parameter called ‘Identifier.’
“xxx”	Indicates a fixed value. Example: Identifier := “abc” meaning value “abc” is assigned to a parameter named ‘Identifier.’
= or ==	A logical condition to indicate an item on the left is equal to an item on the right.
<	A logical condition to indicate an item on the left is less than the item on the right.
>	A logical condition to indicate an item on the left is greater than the item on the right.
<> or != or ≠	A logical condition to indicate an item on the left is not equal to an item on the right.
>>	A semantic condition with the meaning “newer”
<<	A semantic condition with the meaning “older”
&&	Logical “AND”
||	Logical “OR”
<=	A logical condition to indicate an item on the left is equal to or less than the item on the right.
>=	A logical condition to indicate an item on the left is equal to or greater than the item on the right.
for sequence-description actions endfor	This construct allows the execution of a sequence of actions in a loop within one transition. The sequence-description shall describe the loop variable and the number of repetitions. The “endfor” can be omitted.

If (condition)
actions
else
actions
endif
	This construct allows the execution of alternative actions depending on some condition (which might be the value of some identifier or the outcome of a previous action) within one transition.
	The parts beginning with “else” can be omitted if there is no action if the condition is not fulfilled.
	The “endif” can be omitted.

// comment /* comment */ Note comment	These constructs are used to distinguish between sentences to be executed and explanatory descriptions (comment).

The conventions used for services in the state machines are shown in Table 9.

Table 9 – Conventions for services used in state machines

Convention	Meaning
xxx.req ()	Request according to ISO/IEC 7498-1 Invokes the service and passes any required parameter Parameters are omitted and listed within the state machine description
xxx.ind ()	Indication according to ISO/IEC 7498-1 Advices the activation of a requested service
xxx.rsp (+), xxx.cnf (+)	Response according to ISO/IEC 7498-1 May positively acknowledge or complete an action previously invoked by a request primitive Confirmation according to ISO/IEC 7498-1 Primitive returned to the requestor to positively acknowledge or complete an action previously invoked by a request primitive Parameters are omitted and listed within the state machine description
xxx.rsp (-), xxx.cnf (-)	Response according to ISO/IEC 7498-1 May negatively acknowledge or complete an action previously invoked by a request primitive Confirmation according to ISO/IEC 7498-1 Primitive returned to the requestor to negatively acknowledge or complete an action previously invoked by a request primitive Parameters are omitted and listed within the state machine description
xxx_req (), xxx_cnf (+), xxx_cnf (-)	This construct is used for the description of local and confirmed services, which are transferred between state machines Parameters are listed within the state machine description and can be delivered optionally inside brackets
xxx_ind ()	This construct is used for the description of local services and indicates the reception of this service Parameters are listed within the state machine description and can be delivered optionally inside brackets
xxx ()	This construct is used for the description of state machine internal functions (local) Parameters are listed within the state machine description and can be delivered optionally inside brackets
(), (+), (-)	Replacement of the associated parameter list, described in the primitives definition Exception: Local services and functions can deliver parameters inside the brackets If the result is not necessary for further executions, the “+” and “-“ signs can be omitted

Readers are strongly recommended to refer to the clauses for the AREP and CREP attribute definitions, the local functions and the FAL-PDU definitions to understand protocol machines. It is assumed that readers have sufficient knowledge of these definitions and they are used subsequently without further explanations.

In addition the following description elements are used:

Wildcard in names
name_XXX: “XXX” is used as wildcard string for all names beginning with “name”.

others: is used as wildcard string for all events which can occur in this state, excluding the explicitly stated events.

The typical use of a wildcard is an Event. In this context there are as many state transitions as possible events for this wildcard exists.

Wildcard in state names
“ANY” is used as a wildcard string for a current state to indicate all states of this state table.

“SAME” is used as wildcard string for a next state to indicate that the transition remains in the same state. Only to be used in combination with “ANY”.

Empty Events
An empty event means that it has always happened and therefore the transition depends only on the condition.

Completeness of the conditions
If an event has one or more conditions and those conditions do not cover the entire scope, then there is an implicit transition for those uncovered scope that has the ignore action and stays in the same state.

Conditional Macro
<CONDITIONAL-MACRO-NAME>

<

CONDITION1: macrobody1

CONDITION2: macrobody2

…

>

CONDITION1, CONDITION2, etc. define all possible cases for the conditional macro. The “CONDITIONAL-MACRO-NAME” acts as placeholder for the “macrocode” depending on the result of the condition.

Replacement Macro
<REPLACEMENT-MACRO-NAME>

<

XXX= Name1 : macrobody1

XXX= Name2 : macrobody2

…

>

Name1, Name2, etc. define all possible cases for the replacement macro. The “REPLACEMENT-MACRO-NAME” acts as placeholder for the “macrocode” depending on the value of the current use of the wildcard XXX.

EXAMPLE

<SERVICE_REQ_PARA>

<

XXX=Read: Para1, Para2

XXX=Write: Para1, Para2, Para3

>

when called in

MSAB_XXX.req(<SERVICE_REQ_PARA>)

will result in

MSAB_Read.req(Para1, Para2) or MSAB_Write.req(Para1, Para2, Para3)

4 Application layer protocol specification for common protocols

4.1 FAL syntax description

4.1.1 DLPDU abstract syntax reference

4.1.1.1 General

Table 10, Table 11 and Table 12 give an outline of the abstract syntax of the DLPDU according to IEEE Std 802.3, IEEE Std 802.11 and IEEE Std 802.15.1.

DLPDUs are generated by end station components (see 4.12.3), transported by bridge components, and received by end station components. Rules for the transmission of frames through bridges are described in 4.12.4.

4.1.1.2 IEEE Std 802.3

The encoding and decoding of the fields in Table 10 shall be according to IEEE Std 802.3 for the DLPDU.

Table 10 – IEEE Std 802.3 DLPDU syntax

DLPDU name	DLPDU structure
DLPDU	Preamble [a], StartFrameDelimiter, DestinationAddress, SourceAddress, DLSDU [b], DLPDU_Padding* [c], FrameCheckSequence
DLSDU	TAAPDU ^ NTAAPDU
TAAPDU	VLAN [d], [FRER] [e], LT, SAPDU ^ FIDAPDU
NTAAPDU	[VLAN] [f], [FRER], LT, SAPDU ^ FIDAPDU
SAPDU	UDP-RTC-PDU ^ UDP-RTA-PDU ^ CL-RPC-PDU ^ LLDP-PDU [g] ^ ICMP-PDU
FIDAPDU	FrameID, RTC-PDU ^ RTA-PDU ^ DCP-PDU ^ PTCP-PDU [g] ^ FRAG-PDU
VLAN	LT(=0x8100), TCI
FRER	LT(=0xF1C1), RTI
NOTE According to IEEE Std 802.3 the DLPDUs have a minimum length of 64 octets (excluded Preamble, Start Frame Delimiter).	
[a] IEEE Std 802.3 defines this field with seven octets. This document defines this field either with seven octets or one octet. [b] The minimum DLSDU size is 2 octets. [c] The number of padding octets shall be in the range of 0..46 depending on the DLSDU size. See the minimum frame size in IEEE Std 802.3 [d] The VLAN field together with the DestinationAddress identifies a stream. [e] The FRER field together with VLAN and DestinationAddress identifies a redundant transmitted stream. [f] The VLAN field of DLPDU containing an RTC-PDU shall be encoded by the sender. The decoder shall accept DLPDUs with or without VLAN field. One VLAN shall be supported, more VLANs may be supported. RT_CLASS_3: The VLAN field shall be omitted by the sender. The RED_RELAY should discard RT_CLASS_3 frames with VLAN. [g] The VLAN field shall be omitted except for the PTCP-AnnouncePDU.	

4.1.1.3 IEEE Std 802.11

The encoding and decoding of the fields in Table 11 shall be according to IEEE Std 802.11 for the DLPDU.

Table 11 – IEEE Std 802.11 DLPDU syntax

DLPDU name	DLPDU structure
DLPDU	DLPDU_1 ^ DLPDU_2 ^ DLPDU_3 ^ DLPDU_4
DLPDU_1	FrameControl, DestinationAddress, SourceAddress, BSSID, SequenceControl, [QoSControl] [a], DLSDU, DLPDU_Padding* [b], FrameCheckSequence
DLPDU_2	FrameControl, DestinationAddress, BSSID, SourceAddress, SequenceControl, [QoSControl] [a], DLSDU, DLPDU_Padding* [b], FrameCheckSequence
DLPDU_3	FrameControl, BSSID, SourceAddress, DestinationAddress, SequenceControl, [QoSControl], DLSDU, DLPDU_Padding* [b], FrameCheckSequence
DLPDU_4	FrameControl, ReceiverAddress, TransmitterAddress, DestinationAddress, SequenceControl, SourceAddress, [QoSControl], DLSDU, DLPDU_Padding* [b], FrameCheckSequence
DLSDU	DSAP(=0xaa), SSAP(=0xaa), CTRL(=0x03), OUI(=0), TAAPDU ^ NTAAPDU
TAAPDU	VLAN, LT, SAPDU ^ FIDAPDU
NTAAPDU	[VLAN] [a], LT, SAPDU ^ FIDAPDU
SAPDU	UDP-RTC-PDU ^ UDP-RTA-PDU ^ CL-RPC-PDU ^ LLDP-PDU ^ ICMP-PDU
FIDAPDU	FrameID, RTC-PDU ^ RTA-PDU ^ DCP-PDU ^ PTCP-PDU
VLAN	LT(=0x8100), TCI
FRER	LT(=0xF1C1), RTI

NOTE 1 For definition of FrameControl, see IEEE Std 802.11.

NOTE 2 For definition of BSSID, see IEEE Std 802.11.

NOTE 3 For definition of SequenceControl, see IEEE Std 802.11.

NOTE 4 For definition of QoSControl, see IEEE Std 802.11.

NOTE 5 For definition of ReceiverAddress, see IEEE Std 802.11.

NOTE 6 For definition of TransmitterAddress, see IEEE Std 802.11.

NOTE 7 IEEE Std 802.11 supports the use of VLANs in conjunction with FrameClassifier Type 2. For definition of FrameClassifier, see IEEE Std 802.11.

NOTE 8 For definition of DSAP, see IEEE Std 802.11.

NOTE 9 For definition of SSAP, see IEEE Std 802.11.

NOTE 10 For definition of CTRL, see IEEE Std 802.11.

NOTE 11 For definition of OUI in this context, see IEEE Std 802.11.

[a] The VLAN field of a RTC DLPDU should be encoded by the sender. The decoder shall accept DLPDUs with or without VLAN fields.

[b] The number of padding octets shall be in the range of 0...46 depending on the DLSDU size. See the minimum frame size in IEEE Std 802.11.

4.1.1.4 IEEE Std 802.15.1

The encoding and decoding of the fields in Table 12 shall be according to IEEE Std 802.15.1 for the DLPDU.

Table 12 – IEEE Std 802.15.1 DLPDU syntax

DLPDU name	DLPDU structure
DLPDUHEADER	Access Code, Packet Header, Payload Header, L2CAP Header, BNEPType(0)
DLPDU	DLPDUHEADER, DestinationAddress, SourceAddress, DLSDU [a], FrameCheckSequence
DLSDU	TAAPDU ^ NTAAPDU
TAAPDU	VLAN, LT, SAPDU ^ FIDAPDU
NTAAPDU	[VLAN] [b], LT, SAPDU ^ FIDAPDU
SAPDU	UDP-RTC-PDU ^ UDP-RTA-PDU ^ CL-RPC-PDU ^ LLDP-PDU [c] ^ ICMP-PDU
FIDAPDU	FrameID, RTC-PDU ^ RTA-PDU ^ DCP-PDU ^ PTCP-PDU [c]
VLAN	LT(=0x8100), TCI
FRER	LT(=0xF1C1), RTI
NOTE 1 For definition of Access Code, see IEEE Std 802.15.1. NOTE 2 For definition of Packet Header, see IEEE Std 802.15.1. NOTE 3 For definition of Payload Header, see IEEE Std 802.15.1. NOTE 4 For definition of L2CAP Header, see IEEE Std 802.15.1. NOTE 5 For definition of BNEPType, see IEEE Std 802.15.1.	
[a] The minimum DLSDU size is 2 octets. [b] The VLAN field of a RTC DLPDU should be encoded by the sender. The decoder shall accept DLPDUs with or without VLAN fields. [c] The VLAN field should be omitted.	

4.1.2 Data types

4.1.2.1 Notation for the Boolean type

```
Boolean ::= BOOLEAN          -- TRUE if the value is non-zero.
                             -- FALSE if the value is zero.
```

4.1.2.2 Notation for the Integer type

```
Integer8 ::= INTEGER  (-128..+127)        -- range -2^7 ≤ I ≤ 2^7-1
Integer16 ::= INTEGER  (-32 768..+32 767) -- range -2^15 ≤ I ≤ 2^15-1
Integer32 ::= INTEGER                     -- range -2^31 ≤ I ≤ 2^31-1
Integer64 ::= INTEGER                     -- range -2^63 ≤ I ≤ 2^63-1
```

4.1.2.3 Notation for the Unsigned type

```
Unsigned8 ::= INTEGER  (0..255)          -- range 0 ≤ I ≤ 2^8-1
Unsigned16 ::= INTEGER  (0..65 535)      -- range 0 ≤ I ≤ 2^16-1
Unsigned32 ::= INTEGER                   -- range 0 ≤ I ≤ 2^32-1
Unsigned64 ::= INTEGER                   -- range 0 ≤ I ≤ 2^64-1
```

4.1.2.4 Notation for the Floating Point type

```
Float32 ::= BIT STRING  SIZE (4)         -- ISO/IEC/IEEE 60559 Single precision
Float64 ::= BIT STRING  SIZE (8)         -- ISO/IEC/IEEE 60559 Double precision
```

4.1.2.5 Notation for the OctetString type

```
OctetString ::= OCTET STRING             -- For generic use.
                                         If used for characters, the coding rules for
                                         VisibleString apply.
```

4.1.2.6 Notation for VisibleString type

```
VisibleString ::= VISIBLE STRING         -- ISO/IEC 646 – International Reference
                                         Version without the “del” (coding 0x7F)
                                         character
```

4.1.2.7 Notation for UnicodeString8 type

```
UnicodeString8 ::= UNICODE STRING        -- UTF-8 variable-width encoding that can
                                         represent every character defined by
                                         ISO/IEC 10646
```

4.1.2.8 Notation for TimeStamp type

```
TimeStamp ::= OCTET STRING  SIZE (12)    -- IEEE Std 802.1AS derived time format
```

4.1.2.9 Notation for TimeStampDifference type

```
TimeStampDifference ::= OCTET STRING  SIZE (12)   -- IEEE Std 802.1AS derived time
                                                  format
```

4.1.2.10 Notation for TimeStampDifferenceShort type

```
TimeStampDifferenceShort ::= OCTET STRING  SIZE (8)   -- IEEE Std 802.1AS derived
                                                      time format
```

4.2 Transfer syntax

4.2.1 Coding of basic data types

4.2.1.1 General encoding

– The encoding of values shall be big endian if not explicitly stated otherwise.

4.2.1.2 Encoding of a Boolean value

– The encoding of a Boolean value shall be primitive. The ContentsOctets shall consist of a single octet.

– If the Boolean value is FALSE, the ContentsOctets shall be 0 (zero). If the Boolean value is TRUE, the ContentsOctets shall not be 0 (zero).

4.2.1.3 Encoding of an Integer value

– The encoding of a fixed-length Integer value of Integer8, Integer16, Integer32 and Integer64 types shall be primitive and the ContentsOctets shall consist of exactly one, two, four, or eight octets, respectively.

– The ContentsOctets shall be a two's complement binary number equal to the integer value and consist of bits 7 to 0 of the first octet, followed by bits 7 to 0 of the second octet, followed by bits 7 to 0 of each octet in turn up to and including the last octet of the ContentsOctets.

NOTE The value of a two's complement binary number is derived by numbering the bits in the ContentsOctets, starting with bit 0 of the last octet as bit zero and ending the numbering with bit 7 of the first octet. Each bit is assigned a numerical value of 2^N, where N is its position in the above numbering sequence. The value of the two's complement binary number is obtained by adding the numerical values assigned to each bit for those bits which are set to one, excluding bit 7 of the first octet and then reducing this value by the numerical value assigned to bit 7 of the first octet if that bit is set to one.

4.2.1.4 Encoding of an Unsigned value

– The encoding of a fixed-length Unsigned value of Unsigned8, Unsigned16, Unsigned32 and Unsigned64 types shall be primitive and the ContentsOctets shall consist of exactly one, two, four, or eight octets, respectively.

– The ContentsOctets shall be a binary number equal to the Unsigned value and consist of bits 7 to 0 of the first octet, followed by bits 7 to 0 of the second octet, followed by bits 7 to 0 of each octet in turn up to and including the last octet of the ContentsOctets.

NOTE The value of a binary number is derived by numbering the bits in the ContentsOctets, starting with bit 0 of the last octet as bit zero and ending the numbering with bit 7 of the first octet. Each bit is assigned a numerical value of 2^N, where N is its position in the above numbering sequence. The value of the binary number is obtained by adding the numerical values assigned to each bit for those bits which are set to one.

4.2.1.5 Encoding of a Floating-Point value

– The encoding of a fixed-length floating-point value of Float32 and Float64 types shall be primitive and the ContentsOctets shall consist of exactly four or eight octets, respectively.

– The ContentsOctets shall contain floating-point values defined in conformance with ISO/IEC/IEEE 60559.
For Float32 the sign is encoded in bit 7 of the first octet. It is followed by the biased exponent starting from bit 6 of the first octet and then the significand starting from bit 6 of the second octet.
For Float64 the sign is encoded in bit 7 of the first octet. It is followed by the biased exponent starting from bit 6 of the first octet and then the significant starting from bit 3 of the second octet.

4.2.1.6 Encoding of a STRING value

– The encoding of a variable length STRING value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be a sequence of octets. The leftmost string element is encoded in the first octet, followed by the second octet, followed by each octet in turn up to and including the last octet as rightmost of the ContentsOctets.

4.2.1.7 Types needing calendar and timezone knowledge

4.2.1.7.1 Encoding of a BinaryDate value

– The encoding of a BinaryDate value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 17.

bits octets	7 msb	6	5	4	3	2	1	0 lsb	Meaning
1	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	0…59 999 ms
2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
3	0	0	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	0…59 min
4	DST	0	0	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	0…23 hours DST: Daylight saving time
5	day of week 2^{2}	2^{1}	2^{0}	day of month 2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	1…7 day of week 1…31 day of month
6	0	0	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	1…12 months
7	0	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	0 … 50 yeARs 2000 to 2050 51 … 99 yeARs 1951 to 1999

Figure 17 – Coding of the data type BinaryDate

4.2.1.7.2 Encoding of a TimeOfDay with date indication value

– The encoding of a TimeOfDay with date indication value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 18.

bits octets	7 msb	6	5	4	3	2	1	0 lsb	Meaning
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	Number of Milliseconds since midnight
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	Number of days since 1984-01-01 T 00:00 Z only with date indication
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

Figure 18 – Encoding of TimeofDay with date indication value

4.2.1.7.3 Encoding of a TimeOfDay without date indication value

– The encoding of a TimeOfDay without date indication value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 19.

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

Figure 19 – Encoding of TimeofDay without date indication value

4.2.1.7.4 Encoding of a TimeDifference with date indication value

– The encoding of a TimeDifference with date indication value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 20.

bits octets	7 msb	6	5	4	3	2	1	0 lsb	Meaning
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Milliseconds
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	Days
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

Figure 20 – Encoding of TimeDifference with date indication value

4.2.1.7.5 Encoding of a TimeDifference without date indication value

– The encoding of a TimeDifference without date indication value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 21.

bits octets	7 msb	6	5	4	3	2	1	0 lsb	Meaning
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Milliseconds
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

Figure 21 – Encoding of TimeDifference without date indication value

4.2.1.7.6 Encoding of a NetworkTime value

– The encoding of a NetworkTime value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 22.

bits octets	7 msb	6	5	4	3	2	1	0 lsb	Meaning
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Seconds according to IETF RFC 5905
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Fractional portion of seconds in $1 / 2^{32}$ according to IETF RFC 5905
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	Status 2^{0}	

Figure 22 – Encoding of a NetworkTime value

The field Status shall be coded according Figure 22 and Table 13.

Table 13 – Status

Value	Meaning
0	NetworkTime related to the global synchronized time
1	NetworkTime related to the local (arbitrary timescale) time

4.2.1.7.7 Encoding of a NetworkTimeDifference value

– The encoding of a NetworkTimeDifference value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 23.

bits octets	7 msb	6	5	4	3	2	1	0 lsb	Meaning
1	Sign 2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}	Fractional portion of seconds in $1 / 2^{32}$
2	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}	
3	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}	
4	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}	
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

Figure 23 – Encoding of NetworkTimeDifference value

For Sign Formula (1) applies.

$$\text{NetworkTimeDifference} = \text{NetworkTime}_1 \text{ minus } \text{NetworkTime}_2 \quad (1)$$

where

NetworkTimeDifference is the difference. Sign indicates positive if the minuend is greater the subtrahend.

NetworkTime$_1$ is the minuend

NetworkTime$_2$ is the subtrahend

minus is a function using calendar, time zone and leap second information to realize the subtraction

4.2.1.8 Types independent from calendar and timezone

4.2.1.8.1 Encoding of a TimeStamp value

– The encoding of a TimeStamp value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 24.

bits octets	7 msb	6	5	4	3	2	1	0 lsb	Meaning
1	Reserved 2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	Status
2	Reserved 2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	Time source 2^{1}	2^{0}	
3	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}	Seconds
4	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}	
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
9	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Nanoseconds
10	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
11	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
12	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

Figure 24 – Encoding of TimeStamp value

The field Time source shall be coded according Figure 24 and Table 14.

Table 14 – Time source

Value	ITU-T G.781.1 state	Meaning
0	Locked	TimeStamp related to global synchronized (global time timescale) time
1	Holdover	TimeStamp related to local (derived from global time timescale) time
2	Free-run	TimeStamp related to local (arbitrary timescale) time
Other	—	Reserved

4.2.1.8.2 Encoding of a TimeStampDifference value

– The encoding of a TimeStampDifference value shall be primitive.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 25.

bits octets	7 msb	6	5	4	3	2	1	0 lsb	Meaning
1	Sign 2^{15}	Reserved 2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	Status
2	Reserved 2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
3	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}	Seconds
4	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}	
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
9	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Nanoseconds
10	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
11	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
12	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

Figure 25 – Encoding of TimeStampDifference value

For Sign Formula (2) applies.

$$\text{TimeStampDifference} = \text{TimeStamp}_1 - \text{TimeStamp}_2 \quad (2)$$

where

TimeStampDifference is the difference. Sign indicates positive if the minuend is greater the subtrahend.

$TimeStamp_1$ is the minuend

$TimeStamp_2$ is the subtrahend

4.2.1.8.3 Encoding of a TimeStampDifferenceShort value

– The encoding of a TimeStampDifferenceShort value shall be according Integer64.

– There is no Length field; the length is encoded implicitly.

– The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 26.

bits octets	7 msb	6	5	4	3	2	1	0 lsb	Meaning
1	Sign 2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}	
2	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}	
3	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}	
4	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}	Nanoseconds
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	

Figure 26 – Encoding of TimeStampDifferenceShort value

For Sign Formula (3) applies.

$$\text{TimeStampDifferenceShort} = \text{TimeStamp}_1 - \text{TimeStamp}_2 \quad (3)$$

where

TimeStampDifferenceShort	is the difference. Sign indicates positive if the minuend is greater than the subtrahend.
$TimeStamp_1$	is the minuend
$TimeStamp_2$	is the subtrahend

4.2.1.9 Encoding of a NULL value

– The encoding of a NULL value shall be primitive.
– There is no Length field; the length is encoded implicitly.
– The ContentsOctet shall be omitted.

4.2.1.10 Encoding of a NIL value

– The encoding of a NIL value shall be primitive.
– There is no Length field; the length is encoded implicitly.
– Each ContentsOctet shall be set to zero.

4.2.1.11 Encoding of an UUID

– ISO/IEC 9834-8 definition applies.
– The encoding of this data type shall be primitive and the ContentsOctets shall consist of exactly sixteen octets.
– There is no Length field; the length is encoded implicitly.
– For DCE RPC V1.1 the ContentsOctets shall be encoded according to Publication C706 of The Open Group.

4.2.2 Coding section related to common basic fields

4.2.2.1 Overview

The common fields of 4.1.1 are part of the RTC-PDU and RTA-PDU.

4.2.2.2 Coding of the DLPDU field SourceAddress

This field shall be coded as data type OctetString[6] with values according to Table 15 and IEEE Std 802.1Q. It contains a 48-bit universal LAN MAC addresses according to IEEE Std 802.

Table 15 – SourceAddress

PDU	Meaning
RTC-PDU, RTA-PDU, UDP-RTC-PDU, UDP-RTA-PDU, CL-RPC-PDU, ICMP-PDU, DCP-PDU	The interface MAC address is used.
IEC 62439-2 (MRP-PDU), IEEE Std 802.1AB (LLDP-PDU), IEEE Std 802.1AS (gPTP-PDU), PTCP-PDU	IEEE Std 802.1Q-2018, 6.1, IEEE Std 802.1Q-2018, 8.5, IEEE Std 802.1Q-2018, 8.13.2 and IEEE Std 802.1AC define the usage of port MAC address for such protocols.

NOTE 1 The port MAC address is used to avoid wrong learning of the Filtering Data Base for the Default VLAN in mode SVL of the connected devices when ports are in the State BLOCKED for the Default VLAN.

NOTE 2 The port MAC address is named “separate individual MAC address associated with each instance of the MAC Service provided to an LLC Entity” in IEEE Std 802.1Q.

EXAMPLE An interface of a device with n ports needs one interface MAC address and n port MAC addresses.

Table 16 shows additional rules for the special case single port device.

Table 16 – Single port device

Feature	Meaning
PTCP sync master	Both, interface MAC address and port MAC address shall be supported
PTCP sync slave or LineDelay measurement only	Interface MAC address shall be supported and port MAC address should be supported
No PTCP support	Interface MAC address shall be supported and port MAC address may be supported

4.2.2.3 Coding of the DLPDU field DestinationAddress

4.2.2.3.1 Overview

This field shall be coded as data type OctetString[6] with values according to the 48-bit universal LAN MAC addresses of IEEE Std 802.

4.2.2.3.2 DCP-PDUs

For DCP-Multicast-PDUs, this field shall be coded according to Table 17, Table 18, Table 19, Table 20 and Table 21. DCP-Unicast-PDUs shall not use multicast or broadcast addresses.

Table 17 – DCP_MulticastMACAdd for Identify

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-00	With FrameID=0xFEFE used for DCP-Identify-ReqPDU

Table 18 – DCP_MulticastMACAdd for Hello

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-01	With FrameID=0xFEFC used for DCP-Hello-ReqPDU

Table 19 – DCP_MulticastMACAdd range 1

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-02 – 00-00-1F	Reserved for other applications

Table 20 – DCP_MulticastMACAdd range for filterable Identify

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-20 – 00-00-3F	With FrameID=0xFEFE used for filterable DCP-Identify-ReqPDU

Table 21 – DCP_MulticastMACAdd range 2

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-40 – 00-00-FF	Reserved for other applications

4.2.2.3.3 RT_CLASS_x Multicast-PDUs

For RT_CLASS_x Multicast-PDUs, the value shall be set according to Table 22, Table 23, and Table 24.

Table 22 – MulticastMACAdd range 1

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-01-00	Reserved for further multicast addresses within the Type 10 context
01-0E-CF	00-01-01	RT_CLASS_3 destination multicast address
01-0E-CF	00-01-02	RT_CLASS_3 invalid frame multicast address
01-0E-CF	00-01-03 – 00-01-FF	Reserved for further multicast addresses within the Type 10 context

Table 23 – MulticastMACAdd range 2

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-02-00 – 00-02-FF	RT_CLASS_2 destination multicast address

Table 24 – MulticastMACAdd range 3

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-03-00 – 00-03-FF	Reserved for further multicast addresses within the Type 10 context

4.2.2.3.4 PTCP-PDUs

For PTCP-PDUs, the value shall be set according to Table 25, Table 26, Table 27, Table 28, Table 29, and Table 30.

Table 25 – PTCP_MulticastMACAdd range 2

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-04-00	In conjunction with PTCP-AnnouncePDU and FrameID (=0xFF00) used for working clock synchronization
01-0E-CF	00-04-01	Reserved – legacy
01-0E-CF	00-04-02 – 00-04-1E	Reserved
01-0E-CF	00-04-1F	Reserved

Table 26 – PTCP_MulticastMACAdd range 3

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-04-20	In conjunction with PTCP-RTSyncPDU with follow up and FrameID(=0x0020) used for working clock synchronization
01-0E-CF	00-04-21	Reserved – legacy
01-0E-CF	00-04-22 – 00-04-3E	Reserved
01-0E-CF	00-04-3F	Reserved

Table 27 – PTCP_MulticastMACAdd range 4

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-04-40	In conjunction with PTCP-FollowUpPDU and FrameID(=0xFF20) used for working clock synchronization
01-0E-CF	00-04-41	Reserved – legacy
01-0E-CF	00-04-41 – 00-04-5E	Reserved
01-0E-CF	00-04-5F	Reserved

Table 28 – PTCP_MulticastMACAdd range 5

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-04-60 – 00-04-7F	Reserved for other applications

Table 29 – PTCP_MulticastMACAdd range 6

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-04-80	In conjunction with PTCP-RTSyncPDU and FrameID(=0x0080) used for working clock synchronization
01-0E-CF	00-04-81	Reserved – legacy
01-0E-CF	00-04-82 – 00-04-9E	Reserved
01-0E-CF	00-04-9F	Reserved

Table 30 – PTCP_MulticastMACAdd range 7

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-80-C2	00-00-0E	In conjunction with PTCP-DelayReqPDU and FrameID(=0xFF40), PTCP-DelayResPDU with follow up and FrameID(=0xFF41), PTCP-DelayFuResPDU and FrameID(=0xFF42) and PTCP-DelayResPDU without follow up and FrameID (=0xFF43) used for delay measurement

4.2.2.3.5 Other multicast PDUs

Table 31, Table 32, Table 33, and Table 34 show additional Multicast MAC address ranges according to this document.

Table 31 – MulticastMACAdd range 8

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-04-A0 – 00-04-FF	Reserved for other applications

Table 32 – MulticastMACAdd range 9

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-05-00	Reserved for vendor specific usage in conjunction with an MRP ring.

Table 33 – MulticastMACAdd range 10

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-05-01 – FF-FF-FD	Reserved for other applications

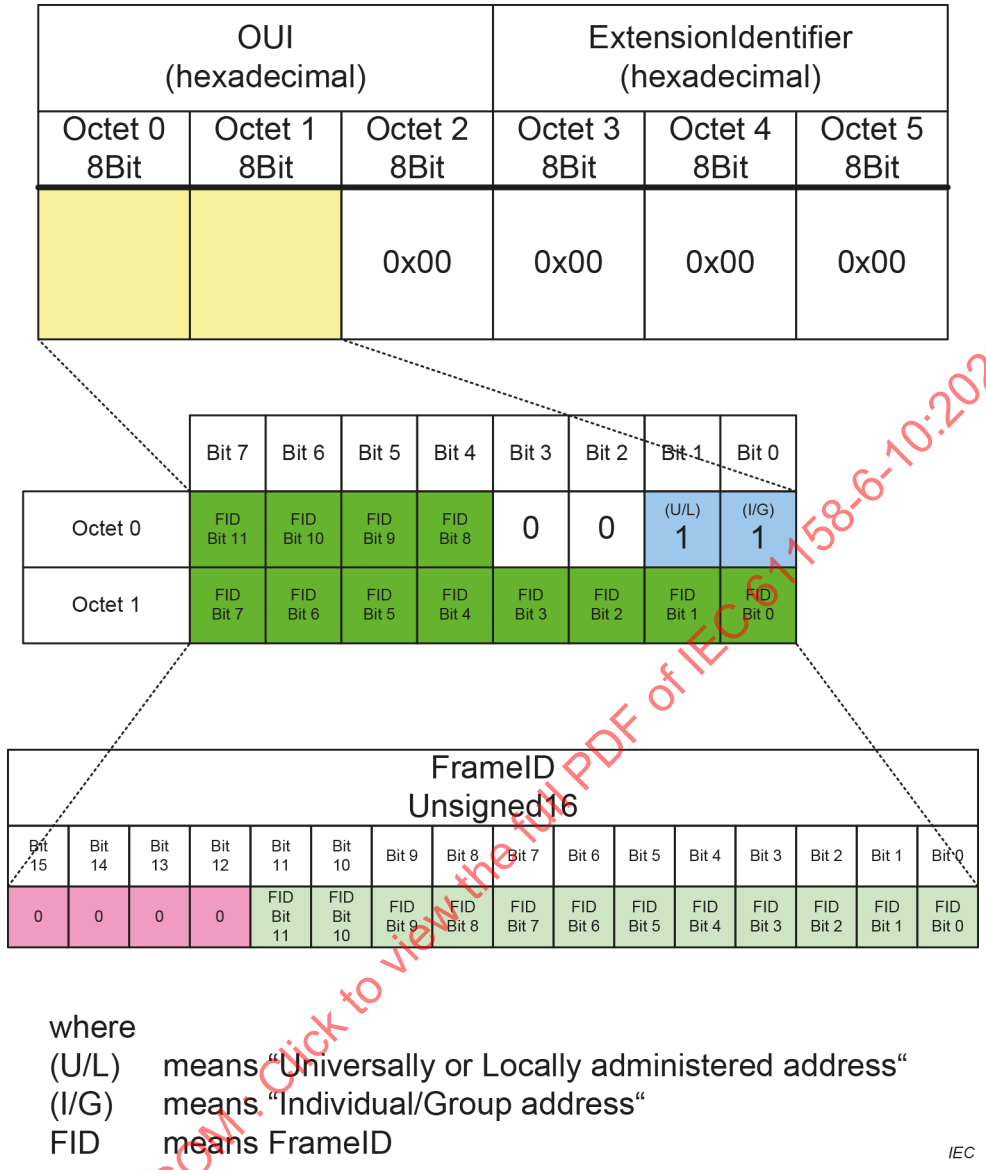
Table 34 – MulticastMACAdd range 11

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	FF-FF-FF	Reserved for net load testing

4.2.2.3.6 RTC-PDUs with FastForwarding

For RTC-PDUs with FastForwarding, this field shall be coded according to Figure 27 as a “Locally administered group address” according to the 48-bit universal LAN MAC addresses of IEEE Std 802.

NOTE Octet 1 contains the Individual/Group Address Bit (lsb).



where
(U/L) means "Universally or Locally administered address"
(I/G) means "Individual/Group address"
FID means FrameID

IEC

Figure 27 – FastForwardingMulticastMACAdd

The field DA is used for the forwarding decision and the field FrameID is used for the CPM decision.

4.2.2.3.7 RTC-PDUs with RT_CLASS_3 destination multicast address

For RTC-PDUs with RT_CLASS_3 destination multicast address, this field shall be coded according to Table 35.

Table 35 – RT_CLASS_3 destination multicast address

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-01-01	RT_CLASS_3 destination multicast address

4.2.2.3.8 RTC-PDUs with RT_CLASS_3 invalid frame multicast address

For RTC-PDUs with RT_CLASS_3 invalid frame multicast address, this field shall be coded according to Table 36.

Table 36 – RT_CLASS_3 invalid frame multicast address

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-01-02	RT_CLASS_3 invalid frame multicast address

4.2.2.3.9 Stream Destination MAC Address

Table 37 shows the two stream categories as a selection criterion for the Destination MAC Address.

Table 37 – Stream categories for RT_CLASS_STREAM

Category	Class	Meaning
Time-aware stream	HIGH and LOW	Stream with a traffic engineered path using a dedicated Destination MAC Address as path identifier
Stream	RT	Stream with a learned path following the spanning tree using either a dedicated Destination MAC Address in case of Multicast communication or the MAC Address of the sink (used as Destination MAC Address) as path identifier

The Stream Destination MAC Address (StreamDA) and TCI.VID identify a path used for a dedicated stream. Thus, the StreamDA needs to be unique together with the TCI.VID in a dedicated NME domain.

Figure 28 shows an optimized format of DestinationAddress used as StreamDA. This format allows the usage of the ID portion of the MAC as index for a table.

An ID portion of the MAC address of 11 Bit covers 2 048 paths, while combined with four VLANs (and IVL) it covers 8 192 paths.

If more independent paths are needed, the ID portion could be extended to e.g. 16 Bit which allows up to 64K or combined with four VLANs, 256K for paths. Moreover, there are eight more bits still available in the ExtensionIdentifier.

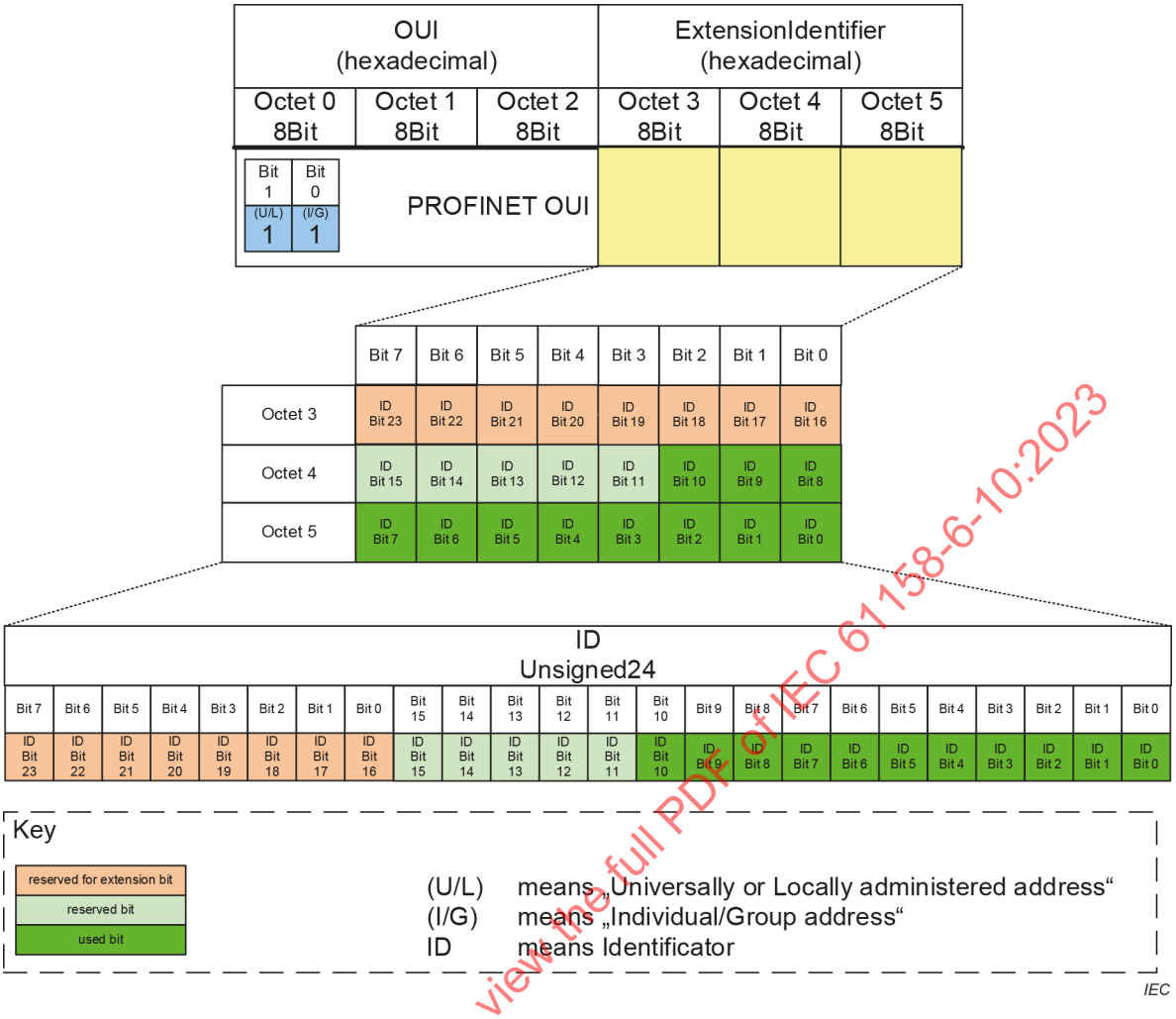


Figure 28 – Stream Destination MAC Address – StreamDA

4.2.2.4 Coding of the field LT

This field shall be coded as data type Unsigned16 with the values according to IEEE Std 802.3. This specification uses the values according to Table 38.

Table 38 – LT (Length/Type)

Value (hexadecimal)	Meaning
0x0800	IP UDP, RPC, SNMP, NetConf, TCP, ICMP, …
0x0806	ARP
0x8100	IEEE Std 802.1Q TCI
0x8892	IEC 61158-x-10 defined protocols RTC, RTA, DCP, PTCP, FRAG, RSI, …
0x88CC	IEEE Std 802.1AB LLDP
0x88E3	IEC 62439-2 MRP

Value (hexadecimal)	Meaning
0x88F7	IEEE Std 802.1AS gPTP
0xF1C1	IEEE Std 802.1CB RTI

4.2.2.5 Coding of the field TCI

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 11: TCI.VID
This field shall be coded according to IEEE Std 802.1Q with the values according to Table 39. This document does not define any VID.

Table 39 – TCI.VID

Value (decimal)	Meaning	
	Non-time-aware system	Time-aware system
0	Default According to IEEE Std 802.1Q definition, No VLAN assigned	Not supported No VLAN assigned
100	IEEE Std 802.1Q definition applies	Non-stream and Stream, class RT Used for DCP, IP, RSI, … and connected “Non-time-aware systems”
101	IEEE Std 802.1Q definition applies	Time-aware stream, class HIGH Used for RT_CLASS_STREAM
102	IEEE Std 802.1Q definition applies	Time-aware stream, class HIGH redundant Used for RT_CLASS_STREAM
103	IEEE Std 802.1Q definition applies	Time-aware stream, class LOW Used for RT_CLASS_STREAM
104	IEEE Std 802.1Q definition applies	Time-aware stream, class LOW redundant Used for RT_CLASS_STREAM
105	IEEE Std 802.1Q definition applies	Non-stream traffic Not used by this specification
106	IEEE Std 802.1Q definition applies	Non-stream traffic Not used by this specification
107	IEEE Std 802.1Q definition applies	Non-stream traffic Not used by this specification
Other	IEEE Std 802.1Q definition applies	Reserved

Bit 12: TCI.DEI
This field shall be coded according to IEEE Std 802.1Q with the values according to Table 40.

Table 40 – TCI.DEI

Value (hexadecimal)	Meaning
0x00	Default Drop Eligible is FALSE
0x01	Reserved Drop Eligible is TRUE

Bit 13 – 15: TCI.PCP
This field shall be coded according to IEEE Std 802.1Q with the values according to Table 377, Table 41, and Table 42.

Table 41 – TCI.PCP for time-aware system

Value (hexadecimal)	Meaning
0	See 4.12.2.2 All protocols which are not explicitly stated 4.12.2.2 shall use traffic class BEH.
1	
2	
3	
4	
5	
6	
7	

Table 42 – TCI.PCP for non-time-aware system

Value (hexadecimal)	Meaning
0	See 4.12.2.3 All protocols which are not explicitly stated in 4.12.2.3 shall use traffic class BEH.
1	
2	
3	
4	
5	
6	
7	

4.2.2.6 Coding of the field RTI

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 15: RTI.SequenceNumber
This field shall be coded according to IEEE Std 802.1CB with the values according to Table 43.

Table 43 – RTI.SequenceNumber

Value (hexadecimal)	Meaning	
	Non-time-aware system	Time-aware system
0x0000 – 0xFFFF	—	Per stream: Each time-aware stream defines its own number space for the SequenceNumber. The PPM starts with the value one.

Bit 16 – 31: RTI.Reserved
This field shall be coded according to IEEE Std 802.1CB with the values according to Table 44.

Table 44 – RTI.Reserved

Value (decimal)	Meaning	
	Non-time-aware system	Time-aware system
Other	Reserved	Reserved

4.2.2.7 Coding of the field FrameID

This field shall be coded as data type Unsigned16 with the values according to Table 45, Table 46, Table 47, Table 48, Table 49, Table 50, Table 51, Table 52, Table 53, Table 54, Table 55, Table 56, Table 57, Table 58 and Table 59. This field identifies the structure and the type of the APDU.

Table 45 – FrameID range 1

Value (hexadecimal)	Meaning	Use
0x0000 – 0x001F	Reserved	Reserved
0x0020	PTCP-RTSyncPDU (with follow up)	Precision transparent clock protocol synchronization with follow up Working Clock for Send Clock and Phase Synchronization
0x0021 – 0x007F	Reserved	Reserved

Table 46 – FrameID range 2

Value (hexadecimal)	Meaning	Use
0x0080	PTCP-RTSyncPDU	Precision transparent clock protocol synchronization without follow up Working Clock for Send Clock and Phase Synchronization
0x0081 – 0x009F	Reserved	—
0x00A0 – 0x00FF	Reserved	—

Table 47 – FrameID range 3a

Value (hexadecimal)	Meaning	Use
0x0100 – 0x06FF	Dedicated to communication class RT_CLASS_3 (RED) unicast and multicast	RED, non-redundant, normal or DFP [a]
0x0700 – 0x0FFF	Dedicated to communication class RT_CLASS_3 (RED) unicast and multicast	RED, redundant [b], normal or DFP

[a] A network monitor can sort “normal” and “DFP” frames by using the first SFCRC16 of the frame.

[b] In this case two FrameIDs (for example 0xXXX0 and 0xXXX1) are used for one CR. One for each direction in which the frame travels the ring. The pair shall be identified ignoring the least significant bit of the FrameID.

Table 48 – FrameID range 3b

Value (hexadecimal)	Meaning	Use
0x1000 – 0x1FFF	RT_CLASS_STREAM Dedicated to Streams (Input CRs or Output CRs)	Used for Time-aware streams, class HIGH and LOW in a time-aware system
0x2000 – 0x2FFF	RT_CLASS_STREAM Dedicated to Streams (Input CRs or Output CRs)	Used for Time-aware streams, class HIGH and LOW in a time-aware system, with security
0x3000 – 0x37FF	RT_CLASS_STREAM Dedicated to Streams	Reserved
0x3800 – 0x3BFF	RT_CLASS_STREAM Dedicated to Streams (Multicast CRs)	Used for Time-aware streams, class HIGH and LOW in a time-aware system, with security
0x3C00 – 0x3FFF	RT_CLASS_STREAM Dedicated to Streams (Multicast CRs)	Used for Time-aware streams, class HIGH and LOW in a time-aware system
0x4000 – 0x47FF	Reserved	—

Table 49 – FrameID range 4

Value (hexadecimal)	Meaning	Use
0x4800 – 0x4FFF	Reserved	—
0x5000 – 0x57FF	Reserved	—
0x5800 – 0x5FFF	Reserved	—
0x6000 – 0x67FF	Reserved	—

Table 50 – FrameID range 5

Value (hexadecimal)	Meaning	Use
0x6800 – 0x6FFF	Reserved	—
0x7000 – 0x77FF	Reserved	—
0x7800 – 0x7FFF	Reserved	—

Table 51 shows that both RT_CLASS_1 and RT_CLASS_STREAM (Streams, class RT) share a common number range. If the IO Controller is inside a NME domain and the IO Device is outside a NME domain, then the IO Controller uses RT_CLASS_STREAM (Streams, class RT), but states RT_CLASS_1 in the Application Relation to the IO Device.

Table 51 – FrameID range 6

Value (hexadecimal)	Meaning	Use
0x8000 – 0x8FFF	Dedicated to communication class RT_CLASS_1 (GREEN) and RT_CLASS_STREAM unicast	Used for Streams, class RT and GREEN, non redundant, normal
0x9000 – 0x9FFF	Dedicated to communication class RT_CLASS_1 (GREEN) and RT_CLASS_STREAM unicast	Used for Streams, class RT and GREEN, non redundant, normal, with security
0xA000 – 0xB7FF	Dedicated to communication class RT_CLASS_1 (GREEN) and RT_CLASS_STREAM	Reserved
0xB800 – 0xBBFF	Dedicated to communication class RT_CLASS_1 (GREEN) and RT_CLASS_STREAM multicast	Used for Streams, class RT and GREEN, non redundant, normal, with security
0xBC00 – 0xBFFF	Dedicated to communication class RT_CLASS_1 (GREEN) and RT_CLASS_STREAM multicast	Used for Streams, class RT and GREEN, non redundant, normal

Table 52 – FrameID range 7

Value (hexadecimal)	Meaning	Use
0xC000 – 0xCFFF	Dedicated to communication class RT_CLASS_UDP unicast	RT_CLASS_UDP
0xD000 – 0xDFFF	Dedicated to communication class RT_CLASS_UDP unicast	RT_CLASS_UDP with security
0xE000 – 0xF3FF	Dedicated to communication class RT_CLASS_UDP unicast	Reserved
0xF400 – 0xF7FF	Dedicated to communication class RT_CLASS_UDP multicast	RT_CLASS_UDP with security
0xF800 – 0xFBFF	Dedicated to communication class RT_CLASS_UDP multicast	RT_CLASS_UDP

Table 53 – FrameID range 8

Value (hexadecimal)	Meaning	Use
0xFC00	Reserved	—
0xFC01	Alarm_High [a]	RTA_CLASS_1 and RTA_CLASS_UDP
0xFC02 – 0xFC40	Reserved	—
0xFC41	Alarm_High [a]	RTA_CLASS_1 and RTA_CLASS_UDP with security.
0xFC42 – 0xFDFF	Reserved	—
0xFE00	Reserved	—
0xFE01	Alarm_Low [a]	RTA_CLASS_1 and RTA_CLASS_UDP
0xFE02	FREQ-RTA-PDU, FRSP-RTA-PDU, ACK-RTA-PDU, ERR-RTA-PDU	Remote Service Interface

Value (hexadecimal)	Meaning	Use
0xFE03 – 0xFE40	Reserved	—
0xFE41	Alarm_Low [a]	RTA_CLASS_1 and RTA_CLASS_UDP with security.
0xFE42	FREQ-RTA-PDU, FRSP-RTA-PDU, ACK-RTA-PDU, ERR-RTA-PDU	Remote Service Interface with security.
0xFE43 – 0xFE7F	Reserved	—
[a] The concurrent use of this FrameID for RTA_CLASS_UDP and RTA_CLASS_1 shall be supported.		

Table 54 – FrameID range 9

Value (hexadecimal)	Meaning	Use
0xFE80 – 0xFEFB	Reserved	—
0xFEFC	DCP-Hello-ReqPDU	DCP
0xFEFD	DCP-Get-ReqPDU, DCP-Get-ResPDU, DCP-Set-ReqPDU, DCP-Set-ResPDU	DCP
0xFEFE	DCP-Identify-ReqPDU	DCP
0xFEFF	DCP-Identify-ResPDU	DCP

Table 55 – FrameID range 10

Value (hexadecimal)	Meaning	Use
0xFF00	PTCP-AnnouncePDU (working clock)	Precision transparent clock announce protocol Isochronous application, send clock and phase synchronization
0xFF01 – 0xFF1F	Reserved	—
0xFF20	PTCP-FollowUpPDU (working clock)	Send Clock and phase synchronization (PTCP-FollowUpPDU)
0xFF21 – 0xFF3F	Reserved	—
0xFF40	PTCP-DelayReqPDU	For delay measurement
0xFF41	PTCP-DelayResPDU	For delay measurement with follow up
0xFF42	PTCP-DelayFuResPDU	For delay measurement with follow up
0xFF43	PTCP-DelayResPDU	For delay measurement without follow up
0xFF44 – 0xFF5F	Reserved	—

Table 56 – FrameID range 11

Value (hexadecimal)	Meaning	Use
0xFF60 – 0xFF6F	Reserved	—

Table 57 – FrameID range 12

Value (hexadecimal)	Meaning	Use
0xFF70 – 0xFF7F	Reserved	—

Table 58 – FrameID range 13

Value (hexadecimal)	Meaning	Use
0xFF80 – 0xFF8F	FragmentationFrameID See 4.2.2.8	Peer to peer protocol to enable SendClockFactors less than 5 independent of the frame size of the concurrent used protocols.

NOTE The principle of fragmentation is usable with all SendClockFactors.

Table 59 – FrameID range 14

Value (hexadecimal)	Meaning	Use
0xFF90 – 0xFFFF	Reserved	—

4.2.2.8 Coding of the field FragmentationFrameID

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 3: FragmentationFrameID.FragSequence
This field shall be used by the receiver to identify the fragmented frame and shall be coded according to Table 60.

Table 60 – FragmentationFrameID.FragSequence

Value (hexadecimal)	Meaning
0x00 – 0x0F	Frame number incremented by the sender for each frame which is fragmented.

Bit 4 – 15: FragmentationFrameID.Constant
This field shall be coded with the values according to Table 61.

Table 61 – FragmentationFrameID.Constant

Value (hexadecimal)	Meaning
0xFF8	Upper part of the FrameID according to Table 58

4.3 Discovery and basic configuration

4.3.1 DCP syntax description

4.3.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.3.1.2 DCP APDU abstract syntax

Table 62 defines the abstract syntax of the DCP-PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs.

Table 62 – DCP APDU syntax

APDU name	APDU structure
DCP-PDU	DCP-Multicast-PDU ^ DCP-Unicast-PDU
DCP-Multicast-PDU	DCP-Identify-ReqPDU [a] ^ DCP-Hello-ReqPDU
DCP-Unicast-PDU	DCP-Get-ReqPDU ^ DCP-Set-ReqPDU ^ DCP-Get-ResPDU ^ DCP-Set-ResPDU ^ DCP-Identify-ResPDU
DCP-Identify-ReqPDU	DCP-IdentifyFilter-ReqPDU ^ DCP-IdentifyAll-ReqPDU
DCP-IdentifyFilter-ReqPDU	DCP-MC-Header [b], [NameOfStationBlock] ^ [AliasNameBlock], [IdentifyReqBlock] * [c]
DCP-IdentifyAll-ReqPDU	DCP-MC-Header, AllSelectorBlock
DCP-Hello-ReqPDU	DCP-UC-Header, NameOfStationBlockRes, IPParameterBlockRes, DeviceIDBlockRes, [DeviceOptionsBlockRes ^ DeviceVendorBlockRes] [d], DeviceRoleBlockRes, DeviceInitiativeBlockRes
DCP-Identify-ResPDU	DCP-UC-Header, Identify-Response-Payload [f,k]
Identify-Response-Payload	{ IdentifyResBlock* [e], IPParameterBlockRes, DeviceVendorBlockRes, NameOfStationBlockRes, DeviceIDBlockRes, DeviceRoleBlockRes, DeviceOptionsBlockRes, [DeviceInstanceBlockRes], [OEMDeviceIDBlockRes], [RsiPropertiesBlockRes] [h], [DeviceInitiativeBlockRes] [g], [NMEDomainBlockRes] [n], [NMEPrioBlockRes] [j], [NMEParameterUUIDBlockRes] [i], [NMENameBlockRes] [i], [CIMInterfaceBlockRes] [i] } The sorting of the Identify-Respose-Payload should be done in ascending order of options and suboptions.
DCP-Get-ReqPDU	DCP-UC-Header, GetReqBlock*
DCP-Get-ResPDU	DCP-UC-Header, (GetResBlock ^ GetNegResBlock)* [l]
DCP-Set-ReqPDU	DCP-UC-Header, [StartTransactionBlock, BlockQualifier], SetResetReqBlock [m] ^ SetReqBlock*, [StopTransactionBlock, BlockQualifier]
DCP-Set-ResPDU	DCP-UC-Header, (SetResBlock ^ SetNegResBlock)*

[a] Shall be supported in conjunction with interface MAC address as Destination MAC Address.

[b] At least one of the optional blocks shall be used.

[c] The content of the field value of the IdentifyReqBlock of the PDU shall be interpreted as a filter at the receiving instance. All included DataBlocks shall be operated with a logical AND and it shall only be responded with a DCP-Identify-ResPDU if all filter criteria match. There shall be at least one optional block.

[d] This field should be omitted by the sender.

[e] The IdentifyResBlock represents the response to the option and suboption requested in the corresponding IdentifyReqBlock or AliasNameBlock. At maximum one AliasName, if requested, shall be responded.

[f] Each of the Option/Suboption blocks shall only be present once; and may be omitted if empty.

[g] This field shall only be present if the usage of DCP-Hello-ReqPDU is activated.

[h] This field shall be present if RSI is supported.

[i] This field shall be present if a CIM is active at this interface.

[j] This field shall be present if an NME is active at this interface.

[k] Additional manufacturer specific blocks following the specified sorting order may be appended by the sender.

[l] If Options/Suboptions requested by DCP-Get-ReqPDU do not fit into DCP-Get-ResPDU, then, starting with the first non-fitting, all following Options/Suboptions shall be skipped silently by the DCPUCR. The user of DCPUCS shall check whether the requested Options/Suboptions will fit into the response or not.

[m] If StartTransaction is used together with FactoryResetBlock or ResetToFactoryBlock, StopTransaction shall be part of the PDU. StopTransaction may be used without StartTransaction.

[n] This field shall be present if time-aware system is supported.

Table 63 defines structures for substitutions of elements of the APDU structures shown in Table 62.

Table 63 – DCP substitutions

Substitution name	Structure
DCP-MC-Header	ServiceID, ServiceType, Xid, ResponseDelayFactor, DCPDataLength
DCP-UC-Header	ServiceID, ServiceType, Xid, Padding* [a], DCPDataLength [a] Number of Padding octets shall be 2.
IdentifyReqBlock	DeviceRoleBlock ^ DeviceVendorBlock ^ DeviceIDBlock ^ DeviceOptionsBlock ^ OEMDeviceIDBlock ^ MACAddressBlock ^ IPParameterBlock ^ DHCPParameterBlock ^ ManufacturerSpecificParameterBlock ^ NMEDomainBlock ^ NMEPrioBlock ^ NMENameBlock ^ CIMInterfaceBlock
IdentifyResBlock	NameOfStationBlockRes ^ DeviceRoleBlockRes ^ DeviceVendorBlockRes ^ DeviceIDBlockRes ^ DeviceOptionsBlockRes ^ OEMDeviceIDBlockRes ^ MACAddressBlockRes ^ IPParameterBlockRes ^ DHCPParameterBlockRes ^ ManufacturerSpecificParameterBlockRes ^ NMEDomainBlockRes ^ NMEPrioBlockRes ^ NMENameBlockRes ^ CIMInterfaceBlockRes ^ AliasNameBlockRes
GetReqBlock	MACAddressType ^ IPParameterType ^ FullIPSuiteType ^ DeviceVendorType ^ NameOfStationType ^ DeviceIDType ^ DeviceRoleType ^ DeviceOptionsType ^ DeviceInstanceType ^ OEMDeviceIDType ^ StandardGatewayType ^ DHCPParameterType ^ DeviceInitiativeType ^ ManufacturerSpecificParameterType ^ RsiPropertiesType ^ NMEDomainType ^ NMEPrioType ^ NMEParameterUUIDType ^ NMENameType ^ CIMInterfaceType

Substitution name	Structure
GetResBlock	MACAddressBlockRes ^ IPParameterBlockRes ^ FullIPSuiteBlockRes ^ DeviceVendorBlockRes ^ NameOfStationBlockRes ^ DeviceIDBlockRes ^ DeviceRoleBlockRes ^ DeviceOptionsBlockRes ^ DeviceInstanceBlockRes^ OEMDeviceIDBlockRes ^ StandardGatewayBlockRes ^ DHCPParameterBlockRes ^ DeviceInitiativeBlockRes ^ ManufacturerSpecificParameterBlockRes ^ RsiPropertiesBlockRes ^ NMEDomainBlockRes ^ NMEPrioBlockRes ^ NMEParameterUUIDBlockRes ^ NMENameBlockRes ^ CIMInterfaceBlockRes
GetNegResBlock	ControlOption, SuboptionResponse, DCPBlockLength, MACAddressType ^ IPParameterType ^ FullIPSuiteType ^ DeviceVendorType ^ NameOfStationType ^ DeviceIDType ^ DeviceRoleType ^ DeviceOptionsType ^ DeviceInstanceType ^ OEMDeviceIDType ^ StandardGatewayType ^ DHCPParameterType ^ DeviceInitiativeType ^ ManufacturerSpecificParameterType RsiPropertiesType ^ NMEDomainType ^ NMEPrioType ^ NMEParameterUUIDType ^ NMENameType ^ CIMInterfaceType, BlockError, [AddDataValue]
SetReqBlock	IPParameterType ^ FullIPSuiteType ^ NameOfStationType ^ DHCPParameterType ^ SignalType ^ ManufacturerSpecificParameterType ^ NMEDomainType, DCPBlockLength, BlockQualifier, IPParameterValue ^ FullIPSuiteValue ^ NameOfStationValue ^ DHCPParameterValue ^ SignalValue ^ ManufacturerSpecificParameterValue ^ NMEDomainValue Only the adjunctive blocks shall be combined to a SetReqBlock Example: SignalType, DCPBlockLength, BlockQualifier, SignalValue
SetResBlock	ControlOption, SuboptionResponse, DCPBlockLength, IPParameterType ^ FullIPSuiteType ^ NameOfStationType ^ DHCPParameterType ^ StartTransactionType ^ StopTransactionType ^ SignalType ^ FactoryResetType ^ ResetToFactoryType ^ ManufacturerSpecificParameterType ^ NMEDomainType, BlockError(=0), [AddDataValue]
SetNegResBlock	ControlOption, SuboptionResponse, DCPBlockLength, IPParameterType ^ FullIPSuiteType ^ NameOfStationType ^ DHCPParameterType ^ StartTransactionType ^ StopTransactionType ^ SignalType ^ FactoryResetType ^ ResetToFactoryType ^ ManufacturerSpecificParameterType ^ NMEDomainType, BlockError(<>0), [AddDataValue]
SetResetReqBlock	FactoryResetBlock ^ ResetToFactoryBlock, BlockQualifier
AllSelectorType	AllSelectorOption, SuboptionAllSelector
AllSelectorBlock	AllSelectorType, DCPBlockLength
DeviceInitiativeType	DeviceInitiativeOption, SuboptionDeviceInitiative
DeviceInitiativeBlockRes	DeviceInitiativeType, DCPBlockLength, BlockInfo, DeviceInitiativeValue
StartTransactionType	ControlOption, SuboptionStart
StartTransactionBlock	StartTransactionType, DCPBlockLength
StopTransactionType	ControlOption, SuboptionStop
StopTransactionBlock	StopTransactionType, DCPBlockLength
SignalType	ControlOption, SuboptionSignal
FactoryResetType	ControlOption, SuboptionFactoryReset
FactoryResetBlock	FactoryResetType, DCPBlockLength
ResetToFactoryType	ControlOption, SuboptionResetToFactory
ResetToFactoryBlock	ResetToFactoryType, DCPBlockLength
NameOfStationType	DevicePropertiesOption, SuboptionNameOfStation
NameOfStationBlock	NameOfStationType, DCPBlockLength, NameOfStationValue
NameOfStationBlockRes	NameOfStationType, DCPBlockLength, BlockInfo, NameOfStationValue

Substitution name	Structure
AliasNameType	DevicePropertiesOption, SuboptionAliasName
AliasNameBlock	AliasNameType, DCPBlockLength, AliasNameValue
AliasNameBlockRes	AliasNameType, DCPBlockLength, BlockInfo, AliasNameValue
DeviceRoleType	DevicePropertiesOption, SuboptionDeviceRole
DeviceRoleBlock	DeviceRoleType, DCPBlockLength, DeviceRoleValue
DeviceRoleBlockRes	DeviceRoleType, DCPBlockLength, BlockInfo, DeviceRoleValue
DeviceRoleValue	DeviceRoleDetails, Padding
DeviceVendorType	DevicePropertiesOption, SuboptionDeviceVendor
DeviceVendorBlock	DeviceVendorType, DCPBlockLength, DeviceVendorValue
DeviceVendorBlockRes	DeviceVendorType, DCPBlockLength, BlockInfo, DeviceVendorValue
DeviceIDType	DevicePropertiesOption, SuboptionDeviceID
DeviceIDBlock	DeviceIDType, DCPBlockLength, DeviceIDValue
DeviceIDBlockRes	DeviceIDType, DCPBlockLength, BlockInfo, DeviceIDValue
DeviceIDValue	VendorIDHigh, VendorIDLow, DeviceIDHigh, DeviceIDLow
OEMDeviceIDType	DevicePropertiesOption, SuboptionOEMDeviceID
OEMDeviceIDBlock	OEMDeviceIDType, DCPBlockLength, DeviceIDValue
OEMDeviceIDBlockRes	OEMDeviceIDType, DCPBlockLength, BlockInfo, DeviceIDValue
StandardGatewayType	DevicePropertiesOption, SuboptionStandardGateway
StandardGatewayBlock	StandardGatewayType, DCPBlockLength, StandardGatewayValue
StandardGatewayBlockRes	StandardGatewayType, DCPBlockLength, BlockInfo, StandardGatewayValue
DeviceOptionsType	DevicePropertiesOption, SuboptionDeviceOptions
DeviceOptionsBlock	DeviceOptionsType, DCPBlockLength, DeviceOptionsValue
DeviceOptionsBlockRes	DeviceOptionsType, DCPBlockLength, BlockInfo, DeviceOptionsValue
DeviceOptionsValue	(Option [a], Suboption [b])* [c] [a] See Table 75 for the values [b] See Table 76, Table 77, Table 78, Table 79, Table 80, Table 81, Table 82, Table 83, and Table 84 for the values [c] If no entry exists, then omit the DeviceOptionsBlockRes in the DCP-Identify-Res-PDU
DeviceInstanceType	DevicePropertiesOption, SuboptionDeviceInstance
DeviceInstanceBlockRes	DeviceInstanceType, DCPBlockLength, BlockInfo, DeviceInstanceValue
DeviceInstanceValue	InstanceHigh, InstanceLow
NMEDomainType	NMEDomainOption, SuboptionNMEDomain
NMEDomainBlock	NMEDomainType, DCPBlockLength, NMEDomainValue
NMEDomainBlockRes	NMEDomainType, DCPBlockLength, BlockInfo, NMEDomainValue
NMEDomainValue	NMEDomainUUID, NMEDomainName
NMEPrioType	NMEDomainOption, SuboptionNMEPrio
NMEPrioBlock	NMEPrioType, DCPBlockLength
NMEPrioBlockRes	NMEPrioType, DCPBlockLength, BlockInfo, NMEPrio
NMENameType	NMEDomainOption, SuboptionNMEName
NMENameBlock	CIMNMENameType, DCPBlockLength
NMENameBlockRes	CIMNMENameType, DCPBlockLength, BlockInfo, NMENameUUID
NMEParameterUUIDType	NMEDomainOption, SuboptionNMEParameterUUID

Substitution name	Structure
NMEParameterUUIDBlockRes	NMEParameterUUIDType, DCPBlockLength, BlockInfo, NMEParameterUUIDValue
NMEParameterUUIDValue	NMEParameterUUID
CIMInterfaceType	NMEDomainOption, SuboptionCIMInterface
CIMInterfaceBlock	CIMInterfaceType, DCPBlockLength
CIMInterfaceBlockRes	CIMInterfaceType, DCPBlockLength, BlockInfo, CIMVDIValue
MACAddressType	IPOption, SuboptionMACAddress
MACAddressBlock	MACAddressType, DCPBlockLength, MACAddressValue
MACAddressBlockRes	MACAddressType, DCPBlockLength, BlockInfo, MACAddressValue
IPParameterType	IPOption, SuboptionIPParameter
IPParameterBlock	IPParameterType, DCPBlockLength, IPParameterValue
IPParameterBlockRes	IPParameterType, DCPBlockLength, BlockInfo, IPParameterValue
IPParameterValue	IPAddress, Subnetmask, StandardGateway
FullIPSuiteType	IPOption, SuboptionFullIPSuite
FullIPSuiteBlockRes	FullIPSuiteType, DCPBlockLength, BlockInfo, FullIPSuiteValue
FullIPSuiteValue	IPAddress, Subnetmask, StandardGateway, IPAddress[4] [a] [a] This array contains up to four DNS server IP addresses. The value "0.0.0.0" indicates "no address".
DHCPParameterType	DHCPOption, SuboptionDHCP
DHCPParameterBlock	DHCPParameterType, DCPBlockLength, DHCPParameterValue
DHCPParameterBlockRes	DHCPParameterType, DCPBlockLength, BlockInfo, DHCPParameterValue
DHCPParameterValue	SuboptionDHCP, DHCPParameterLength, DHCPParameterData
ManufacturerSpecificParameterType	ManufacturerSpecificOption, SuboptionManufacturerSpecific
ManufacturerSpecificParameterBlock	ManufacturerSpecificParameterType, DCPBlockLength, ManufacturerSpecificParameterValue
ManufacturerSpecificParameterBlockRes	ManufacturerSpecificParameterType, DCPBlockLength, BlockInfo, ManufacturerSpecificParameterValue
ManufacturerSpecificParameterValue	ManufacturerOUI, ManufacturerSpecificString
RsiPropertiesType	DevicePropertiesOption, SuboptionRsiProperties
RsiPropertiesBlockRes	RsiPropertiesType, DCPBlockLength, BlockInfo, RsiPropertiesValue [a] [a] Shall contain the sum of all existing RsiPropertiesValue according to the responding unicast SourceAddress.
CIMVDIValue	VendorIDHigh, VendorIDLow, DeviceIDHigh, DeviceIDLow, InstanceHigh, InstanceLow

4.3.1.3 Coding section related to header fields

4.3.1.3.1 Coding of the field ServiceID

This field shall be coded as data type Unsigned8 and shall contain the values as described in Table 64 and Table 65. They shall be set in the appropriate PDU.

Table 64 – ServiceID

Value (hexadecimal)	Meaning	Allowed destination MAC address
0x00 – 0x02	Reserved	—
0x03	Get	Interface MAC address of the station receiving this message.
0x04	Set	Interface MAC address of the station receiving this message.
0x05	Identify	See Table 65
0x06	Hello	Well defined multicast MAC address associated with this service.
0x07 – 0xFF	Reserved	—

Table 65 – Destination MAC addresses used together with the Identify service

Value	Meaning
Interface MAC address	The Identify service is used to read all data assigned to an identify response from the addressed station.
DCP_MulticastMACAdd for Identify	The Identify service is used by IO controller or an Engineering Tool to find the addressed station.
DCP_MulticastMACAdd range for filterable Identify	The Identify service is used by IO controller to find the addressed station. The to be used address out of this address range is calculated according to Formula (4).

The selector for the DCP_MulticastMACAdd (selecting one entry from the range for filterable requests) shall be calculated according to Formula (4).

$$Selector = MD5(NameOfStation)\ \&\ 0x1F \quad (4)$$

where

Selector is the selector for the destination multicast MAC address

MD5 is the IETF RFC 6151 defined function to create a 128 bit fingerprint

NameOfStation is the variable length input parameter NameOfStation

4.3.1.3.2 Coding of the field ServiceType

4.3.1.3.2.1 Coding of the field ServiceType for request

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0: ServiceType.Selection
This field shall be coded with the values according to Table 66.

Table 66 – ServiceType.Selection

Value (hexadecimal)	Meaning
0x00	Request
0x01	Reserved

Bit 1 – 7: ServiceType.Reserved
This field shall be coded with the values according to Table 67.

Table 67 – ServiceType.Reserved

Value (hexadecimal)	Meaning
0x00	Default
0x01 – 0x7F	Reserved

4.3.1.3.2.2 Coding of the field ServiceType for response

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0: ServiceType.Selection
This field shall be coded with the values according to Table 68.

Table 68 – ServiceType.Selection

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Response

Bit 1: ServiceType.Reserved_1
This field shall be coded with the values according to Table 69.

Table 69 – ServiceType.Reserved_1

Value (hexadecimal)	Meaning
0x00	Default
0x01	Reserved

Bit 2: ServiceType.Response
This field shall be coded with the values according to Table 70.

Table 70 – ServiceType.Response

Value (hexadecimal)	Meaning
0x00	Success Default
0x01	ServiceID not supported Optional, otherwise the request may be ignored.

Bit 3 – 7: ServiceType.Reserved_2
This field shall be coded with the values according to Table 71.

Table 71 – ServiceType.Reserved_2

Value (hexadecimal)	Meaning
0x00	Default
0x01 – 0x1F	Reserved

4.3.1.3.3 Coding of the field Xid

This field shall be coded as data type Unsigned32. It shall contain a transaction identification chosen by the client to associate requests and responses between a client and a server.

4.3.1.3.4 Coding of the field DCPDataLength

This field shall be coded as data type Unsigned16. It shall contain the total length of data followed by the DCP-UC-Header or DCP-MC-Header in octets. The maximum length of DCP Data is 1 432 octets.

4.3.1.3.5 Coding of the field ResponseDelayFactor

4.3.1.3.5.1 General

This field shall be coded as data type Unsigned16 with values according to Table 72. It contains a delay factor that is used by the server to calculate an individual response delay and by the client to calculate the timeout for a response.

Table 72 – ResponseDelayFactor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Allowed value without spread Used by the IO controller for IO device discovery
0x0002 – 0x1900	Allowed values with spread Used by Engineering Tools for device discovery
0x1901 – 0xFFFF	Reserved

4.3.1.3.5.2 Server or responder

The server shall calculate the response delay time (see Table 73) according to the following formulae.

The last two octets of the responder MAC address (see the 48-bit universal LAN MAC addresses of IEEE Std 802) shall be used as random number K if no RNG is available. Octet 6 of the MAC address represents the low-order octet and octet 5 the high-order octet.

$$\text{Spread} = K \text{ MOD } \text{ResponseDelayFactor} \quad (5)$$

where

Spread is the specific spreading factor

K is a random number created by a RNG. If not available, the last two octets of the responder MAC address shall be used as random number

ResponseDelayFactor is the factor for the calculation of an individual response delay

Spread is not equal to zero:

$$\text{Minimum Response Delay} = 10\ \text{ms} \times \text{Spread} \quad (6)$$

where

Minimum Response Delay is the minimum response delay

Spread is the specific spreading factor

Spread equals zero:

$$\text{Minimal Response Delay} = 0\ \text{ms} \quad (7)$$

where

Minimal Response Delay is the minimum response delay

Table 73 – ResponseDelayTime

Value [ms]	Meaning
0	Minimum value The server shall respond immediately
10 – 63 990	Possible values for an individual response delay in 10 ms increments
64 000	Maximum value The server shall wait the calculated response delay before the response

4.3.1.3.5.3 Client or requester

The client shall calculate the response delay timeout (see Table 74) according to the following formulae.

ResponseDelayFactor equals 1:

$$\text{Response delay timeout} = 400\ \text{ms} \quad (8)$$

where

Response delay timeout is the response delay timeout

ResponseDelayFactor is greater than 1:

$$\text{Response delay timeout} = \text{Round}\ (1\ \text{s} + \text{ResponseDelayFactor} \times 10\ \text{ms}) \quad (9)$$

where

Response delay timeout is the response delay timeout

Round is a function that round the value up to the next full second

ResponseDelayFactor is the factor for the calculation of an individual response delay

Table 74 – ResponseDelayTimeout

Value [ms]	Meaning
400	Minimum value The client (for example IO controller) expects the response from the server before this timeout
2 000 – 64 000	Possible timeout values in 1 s increments
65 000	Maximum value The client (for example Engineering Tool) expects the response from the server before this timeout

4.3.1.4 Coding section of block fields

4.3.1.4.1 General

The block fields are parted into option, suboption, block length, block info and value. Every block shall assure Unsigned16 alignment. The added padding octets, with the value zero, shall be counted for the DCPDataLength and shall not be counted for the DCPBlockLength. Table 75 shows the list of available options and Table 76, Table 77, Table 78, Table 79, Table 80, Table 81, Table 82 and Table 83 show the list of available suboptions.

Table 75 – List of options

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	IPOption
0x02	DevicePropertiesOption
0x03	DHCPOption
0x04	Reserved
0x05	ControlOption
0x06	DeviceInitiativeOption
0x07	NMEDomainOption
0x08 – 0x7F	Reserved
0x80 – 0xFE	ManufacturerSpecificOption
0xFF	AllSelectorOption

Table 76 – List of suboptions for option IPOption

Value (hexadecimal)	Meaning	Accessible	Identify filter
0x01	SuboptionMACAddress	Read	—
0x02	SuboptionIPParameter	Read / Write	Yes
0x03	SuboptionFullIPSuite	Read / Write Optional	Yes If accessible
Other	Reserved	—	—

Table 77 – List of suboptions for option DevicePropertiesOption

Value (hexadecimal)	Meaning	Accessible	Identify filter
0x01	SuboptionDeviceVendor	Read	Yes
0x02	SuboptionNameOfStation	Read / Write	Yes
0x03	SuboptionDeviceID	Read	Yes
0x04	SuboptionDeviceRole	Read	Yes
0x05	SuboptionDeviceOptions	Read	Yes
0x06	SuboptionAliasName	Used as filter only	Yes
0x07	SuboptionDeviceInstance	Read Optional	Yes If accessible
0x08	SuboptionOEMDeviceID	Read Optional	Yes If accessible
0x09	SuboptionStandardGateway	Read Optional [a]	Yes If accessible
0x0A	SuboptionRsiProperties	Read Optional [b]	—
Other	Reserved	—	—

[a] Shall be supported if StandardGatewayValue.StandardGateway == 0x01

[b] Shall be supported in conjunction with RSI

Table 78 – List of suboptions for option DHCPOption

Value (hexadecimal)	Meaning	Accessible	Identify filter
See Table 84	SuboptionDHCP	Read / Write Optional	Yes If accessible

Table 79 – List of suboptions for option ControlOption

Value (hexadecimal)	Meaning	Accessible	Identify filter
0x01	SuboptionStart [a]	Write	—
0x02	SuboptionStop [a]	Write	—
0x03	SuboptionSignal	Write	—
0x04	SuboptionResponse	—	—
0x05	SuboptionFactoryReset	Write Optional	—
0x06	SuboptionResetToFactory	Write	—
Other	Reserved	—	—

[a] Shall be ignored by the application if not supported

Table 80 – List of suboptions for option DeviceInitiativeOption

Value (hexadecimal)	Meaning	Accessible	Identify filter
0x01	SuboptionDeviceInitiative	Read	Yes
Other	Reserved	—	—

Table 81 – List of suboptions for option NMEDomainOption

Value (hexadecimal)	Meaning	Accessible	Identify filter
0x01	SuboptionNMEDomain	Read / Write	Yes
0x02	SuboptionNMEPrio	Read	Yes
0x03	SuboptionNMEParameterUUID	Read	—
0x04	SuboptionNMEName	Read	Yes
0x05	SuboptionCIMInterface	Read	Yes
Other	Reserved	—	—

Table 82 – List of suboptions for option AllSelectorOption

Value (hexadecimal)	Meaning	Accessible	Identify filter
0xFF	SuboptionAllSelector	Used as filter only	Yes
Other	Reserved	—	—

Table 83 – List of suboptions for option ManufacturerSpecificOption

Value (hexadecimal)	Meaning	Accessible	Identify filter
0x00 – 0xFF	SuboptionManufacturerSpecific	Manufacturer specific	Manufacturer specific

4.3.1.4.2 Coding section related to IPOption

4.3.1.4.2.1 Coding of the field IPOption

This field shall be coded as data type Unsigned8 according to Table 75.

4.3.1.4.2.2 Coding of the field SuboptionMACAddress

This field shall be coded as data type Unsigned8 according to Table 76.

4.3.1.4.2.3 Coding of the field SuboptionIPParameter

This field shall be coded as data type Unsigned8 according to Table 76.

4.3.1.4.2.4 Coding of the field SuboptionFullIPSuite

This field shall be coded as data type Unsigned8 according to Table 76.

4.3.1.4.3 Coding section related to DevicePropertiesOption

4.3.1.4.3.1 Coding of the field DevicePropertiesOption

This field shall be coded as data type Unsigned8 according to Table 75.

4.3.1.4.3.2 Coding of the field SuboptionDeviceVendor

This field shall be coded as data type Unsigned8 according to Table 77.

4.3.1.4.3.3 Coding of the field SuboptionNameOfStation

This field shall be coded as data type Unsigned8 according to Table 77.

4.3.1.4.3.4 Coding of the field SuboptionDeviceID

This field shall be coded as data type Unsigned8 according to Table 77.

4.3.1.4.3.5 Coding of the field SuboptionDeviceRole

This field shall be coded as data type Unsigned8 according to Table 77.

4.3.1.4.3.6 Coding of the field SuboptionDeviceOptions

This field shall be coded as data type Unsigned8 according to Table 77.

This option/suboption combination should be used to convey a list of the supported options and suboptions which are defined as optional and could be used for all options and suboptions.

4.3.1.4.3.7 Coding of the field SuboptionAliasName

This field shall be coded as data type Unsigned8 according to Table 77.

4.3.1.4.3.8 Coding of the field SuboptionDeviceInstance

This field shall be coded as data type Unsigned8 according to Table 77.

4.3.1.4.3.9 Coding of the field SuboptionOEMDeviceID

This field shall be coded as data type Unsigned8 according to Table 77.

4.3.1.4.3.10 Coding of the field SuboptionStandardGateway

This field shall be coded as data type Unsigned8 according to Table 77.

4.3.1.4.3.11 Coding of the field SuboptionRsiProperties

This field shall be coded as data type Unsigned8 according to Table 77.

4.3.1.4.4 Coding section related to DHCPOption

4.3.1.4.4.1 Coding of the field DHCPOption

This field shall be coded as data type Unsigned8 according to Table 75.

4.3.1.4.4.2 Coding of the field SuboptionDHCP

This field shall be coded as data type Unsigned8. The allowed values shall be according to Table 84.

Table 84 – SuboptionDHCP

Value (decimal)	Description	References	Accessible
0 – 11	Reserved [a]	—	—
12	Host Name [b]	IETF RFC 2132	Read / Write Optional
13 – 42	Reserved [a]	—	—
43	Vendor specific information	IETF RFC 2132	Read / Write Optional
44 – 53	Reserved [a]	—	—
54	Server identifier	IETF RFC 2132	Read / Write Optional
55	Parameter request list [c]	IETF RFC 2132	Read / Write Optional
56 – 59	Reserved [a]	—	—
60	Class identifier	IETF RFC 2132	Read / Write Optional
61	DHCP client identifier	IETF RFC 2132, IETF RFC 4361, and IETF RFC 4363	Read / Write Optional
62 – 80	Reserved [a]	—	—
81	FQDN, Fully Qualified Domain Name [b]	—	Read / Write Optional
82 – 96	Reserved [a]	—	—
97	UUID-based Client (=addressed Station) Identifier	—	Read / Write Optional
98 – 254	Reserved [a]	—	—
255 [d]	See 4.3.1.4.23	—	Read / Write

[a] The reserved options should be used in the same meaning as they are defined in the corresponding IETF RFCs for DHCP.

[b] The use of these suboptions is still not fixed and subject to change in later revisions.

[c] In this option, at least the parameters 1 (subnet mask) and 3 (router) should be requested.

[d] In IETF RFC 2132 defined as “END” but handled internally. Thus this key is overridden as DHCP control in this document.

4.3.1.4.5 Coding section related to ControlOption

4.3.1.4.5.1 Coding of the field ControlOption

This field shall be coded as data type Unsigned8 according to Table 75.

4.3.1.4.5.2 Coding of the field SuboptionStart

This field shall be coded as data type Unsigned8 according to Table 79 and with a DCPBlockLength according to Table 85.

Table 85 – Coding of DCPBlockLength in conjunction with SuboptionStart

Value (hexadecimal)	Meaning
0x0000 – 0x0001	Reserved
0x0002	Default
0x0003 – 0xFFFF	Reserved

There is no additional data for this suboption.

4.3.1.4.5.3 Coding of the field SuboptionStop

This field shall be coded as data type Unsigned8 according to Table 79 and with a DCPBlockLength according to Table 86.

Table 86 – Coding of DCPBlockLength in conjunction with SuboptionStop

Value (hexadecimal)	Meaning
0x0000 – 0x0001	Reserved
0x0002	Default
0x0003 – 0xFFFF	Reserved

There is no additional data for this suboption.

4.3.1.4.5.4 Coding of the field SuboptionSignal

This field shall be coded as data type Unsigned8 according to Table 79 and with a DCPBlockLength according to Table 87.

Table 87 – Coding of DCPBlockLength in conjunction with SuboptionSignal

Value (hexadecimal)	Meaning
0x0000 – 0x0003	Reserved
0x0004	Default
0x0005 – 0xFFFF	Reserved

4.3.1.4.5.5 Coding of the field SuboptionResponse

This field shall be coded as data type Unsigned8 according to Table 79.

4.3.1.4.5.6 Coding of the field SuboptionFactoryReset

This field shall be coded as data type Unsigned8 according to Table 79 and with a DCPBlockLength according to Table 88.

Table 88 – Coding of DCPBlockLength in conjunction with SuboptionFactoryReset

Value (hexadecimal)	Meaning
0x0000 – 0x0001	Reserved
0x0002	Default
0x0003 – 0xFFFF	Reserved

There is no additional data for this suboption.

Table 89 shows the alignment between FactoryReset and ResetToFactory.

Table 89 – Alignment between FactoryReset and ResetToFactory

ResetToFactory modes	Alignment
1	Shall be included when FactoryReset is executed
2	Shall be included when FactoryReset is executed
3	May be included when FactoryReset is executed
4	May be included when FactoryReset is executed
8	May be included when FactoryReset is executed
9	May be included when FactoryReset is executed

Every device should support a possibility to execute FactoryReset without an engineering device. This could be done by use of a switch or by other means.

The execution of FactoryReset should not influence the switching of a node.

4.3.1.4.5.7 Coding of the field SuboptionResetToFactory

This field shall be coded as data type Unsigned8 according to Table 79 and with a DCPBlockLength according to Table 90.

This Suboption offers different modes coded by the BlockQualifier with the meaning shown in Table 91.

Table 90 – Coding of DCPBlockLength in conjunction with SuboptionResetToFactory

Value (hexadecimal)	Meaning
0x0000 – 0x0001	Reserved
0x0002	Default
0x0003 – 0xFFFF	Reserved

There is no additional data for this suboption.

Table 91 shows the behavior of the reset command for the different modes.

Table 91 – Meaning of the different ResetToFactory modes

Affected object	Interface				Device	
	Mode 1	Mode 2	Mode 3	Mode 4	Mode 8	Mode 9
Persistent CIM data SNMP writeable OIDs	NULL	NULL	NULL	NULL	NULL	NULL
Persistent CIM data DCP writeable options/suboptions (Examples are IP-suite or NameOfStation/NameOfInterface)	—	NULL	NULL	NULL	NULL	NULL
Persistent CIM data Records PDev data	—	NULL	NULL	NULL	NULL	NULL
Persistent CIM data CIM including SNMP adjust	—	NULL	NULL	NULL	NULL	NULL
Persistent CIM data Records I&M data	NULL	—	NULL	NULL	NULL	NULL
Persistent CIM data Records Data (not PDev)	NULL	—	NULL	NULL	NULL	NULL
Protected persistent storage Credentials Persistent Security Mode (trust store / key store content)	—	—	—	—	—	—
Persistent engineering data (manufacturer specific)	—	—	Deleted	Deleted	Deleted	Deleted
Persistent firmware storage	—	—	—	—	—	Default value
Device hard or soft reset (temporary no response to DCP requests or Ethernet switching)	—	—	—	Allowed	Allowed	Allowed

4.3.1.4.6 Coding section related to DeviceInitiativeOption

4.3.1.4.6.1 Coding of the field DeviceInitiativeOption

This field shall be coded as data type Unsigned8 according to Table 75.

4.3.1.4.6.2 Coding of the field SuboptionDeviceInitiative

This field shall be coded as data type Unsigned8 according to Table 80 and with a DCPBlockLength according to Table 92.

Table 92 – Coding of DCPBlockLength in conjunction with SuboptionDeviceInitiative

Value (hexadecimal)	Meaning
0x0000 – 0x0003	Reserved
0x0004	Default
0x0005 – 0xFFFF	Reserved

4.3.1.4.7 Coding section related to NMEDomainOption

4.3.1.4.7.1 Coding of the field NMEDomainOption

This field shall be coded as data type Unsigned8 according to Table 75.

4.3.1.4.7.2 Coding of the field SuboptionNMEDomain

This field shall be coded as data type Unsigned8 according to Table 81.

4.3.1.4.7.3 Coding of the field SuboptionNMEPrio

This field shall be coded as data type Unsigned8 according to Table 81.

4.3.1.4.7.4 Coding of the field SuboptionNMEParameterUUID

This field shall be coded as data type Unsigned8 according to Table 81.

4.3.1.4.8 Coding section related to ManufacturerSpecificOption

4.3.1.4.8.1 Coding of the field ManufacturerSpecificOption

This field shall be coded as data type Unsigned8 according to Table 75.

4.3.1.4.8.2 Coding of the field SuboptionManufacturerSpecific

This field shall be coded as data type Unsigned8 according to Table 83.

4.3.1.4.9 Coding section related to AllSelectorOption

4.3.1.4.9.1 Coding of the field AllSelectorOption

This field shall be coded as data type Unsigned8 according to Table 75.

4.3.1.4.9.2 Coding of the field SuboptionAllSelector

This field shall be coded as data type Unsigned8 according to Table 82.

4.3.1.4.10 Coding of the field DCPBlockLength

This field shall be coded as data type Unsigned16 according to Table 93.

Table 93 – Coding of DCPBlockLength

Value (hexadecimal)	Meaning
0x0000 – 0x05FF	Length of the specific data of the suboption without DCPBlockLength itself and without counting padding octets
0x0600 – 0xFFFF	Reserved

4.3.1.4.11 Coding of the field BlockQualifier

This field shall be coded as data type Unsigned16.

Allowed values with the option IPOption, DevicePropertiesOption, DHCPOption and ManufacturerSpecificOption shall be according to Table 94.

Table 94 – BlockQualifier with options IPOption, DevicePropertiesOption, DHCPOption and ManufacturerSpecificOption

Bit	Value (binary)	Meaning
Bit 0	0	Store the value of the option/suboption and its attribute “temporary” in the DCP ASE and use the value subsequently. The attribute “temporary” means that the value shall be reset to its default value after a power off / power on cycle.
	1	Store the value of the option/suboption and its attribute “permanent” in the DCP ASE and use the value subsequently. The attribute “permanent” means that the value shall be available after a power off / power on cycle.
Bit 1 – 15	—	Should be set to zero by the sender and shall not be checked by the receiver.

Each successful DCP set modifies the addressed DCP ASE attributes and, if necessary, the persistent storage of these attributes.

Allowed values with the option ControlOption and suboption SuboptionResetToFactory shall be according to Table 95.

Table 95 – BlockQualifier with option ControlOption and suboption SuboptionResetToFactory

Bit	Value (decimal)	Meaning	Range
Bit 0	—	Should be set to zero by the sender and shall not be checked by the receiver.	—
Bit 1 – 15	0	Reserved	—
	1	Recommended Reset application data Reset data which has been stored permanently in submodules and modules to factory values. “Reset application data” and “Reset communication parameter” are intended to be non-overlapping.	Interface
	2	Mandatory Reset communication parameter All parameters active for the interface or the ports and the ARs shall be set to the default values and reset, if permanently stored. Sets the addressed communication interface of a device to a state similar to “out of the box” state. “Reset application data” and “Reset communication parameter” are intended to be non-overlapping.	
	3	Recommended Reset engineering parameter Reset engineering parameters which have been stored permanently to factory values.	
	4	Recommended Reset all stored data Reset all stored data to factory default values.	
	5 – 7	Reserved	
	8	Recommended Reset device	Device

Bit	Value (decimal)	Meaning	Range
		Reset all stored data to factory values. Reset the communication parameters of all interfaces of the device and all parameters of the device. Sets the device to a state similar to “out of the box” state.	
	9	Optional Reset and restore data Reset installed software revisions to factory images.	
	10 – 15	Reserved	
	Other	Reserved	—

Allowed values with the option NMEDomainOption shall be according to Table 96.

Table 96 – BlockQualifier with option NMEDomainOption

Bit	Value (binary)	Meaning
Bit 0	0	Reserved
	1	Store the value of the option/suboption and its attribute “permanent” in the DCP ASE and use the value subsequently. The attribute “permanent” means that the value shall be available after a power off / power on cycle.
Bit 1 – 15	—	Should be set to zero by the sender and shall not be checked by the receiver.

Allowed values with all other options shall be according to Table 97.

Table 97 – BlockQualifier with other options

Bit	Value (binary)	Meaning
Bit 0 – 15	—	Should be set to zero by the sender and shall not be checked by the receiver.

4.3.1.4.12 Coding of the field BlockError

This field shall be coded as data type Unsigned8 and shall be set according to Table 98.

Table 98 – BlockError

Value (hexadecimal)	Meaning	Usage
0x00	No error	Positive response Parameter delivered and accepted
0x01	Option not supported	If optional option is not supported
0x02	Suboption not supported or no DataSet available or BlockQualifier not supported	If optional suboption is not supported
0x03	Suboption not set	If suboption could not be set – for local reasons – wrong semantics – write access not defined
0x04	Resource error	If there is a temporary resource error within the server
0x05	By local reasons, SET or GET not possible	If Set service is not possible for local reasons like – ProtectionOn := TRUE – RejectDCPSetRequests activated by an established AR
0x06	In operation, SET or GET not possible	If Set service is not possible because of application operation, like – AR established – local rules disallow reset
0x07 – 0xFF	Reserved	Shall not be used.

4.3.1.4.13 Coding of the field BlockInfo

This field shall be coded as data type Unsigned16. The allowed values shall be set according to Table 99, Table 100, Table 101 and Table 102.

Table 99 – BlockInfo for SuboptionIPParameter

Bit	Meaning
Bit 0 – 1	See Table 100
Bit 2 – 6	Reserved
Bit 7	See Table 101
Bit 8 – 15	Reserved

Table 100 – Bit 1 and Bit 0 of BlockInfo for SuboptionIPParameter

Bit 1	Bit 0	Meaning for SuboptionIPParameter
0	0	No IP parameter The IP suite (fields IPAddress, Subnetmask and StandardGateway) is deleted. The IP stack is stopped.
0	1	IP parameter set The IP suite (fields IPAddress, Subnetmask and StandardGateway) is available. The IP stack is started or running.
1	0	IP parameter set via DHCP The IP suite (fields IPAddress, Subnetmask and StandardGateway) is available and was set using DHCP. The IP stack is started or running.
1	1	Reserved

Table 101 – Bit 7 of BlockInfo for SuboptionIPParameter

Bit 7	Meaning for SuboptionIPParameter
0	No IP address conflict detected [a] IP address active
1	IP address conflict detected [a] IP address not active
[a] An optional locally defined test shows that the requested IP address is already in use by a different device. The device is not able to activate the delivered IP address.	

Table 102 – BlockInfo for all other suboptions

Bit	Meaning
Bit 0 – 15	Reserved

4.3.1.4.14 Coding of the field DeviceInitiativeValue

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0: DeviceInitiativeValue.Hello
This field shall be set according to Table 103.

Table 103 – DeviceInitiativeValue

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Device does not issue a DCP-Hello-ReqPDU after power on.
0x01	ON	Device issues a DCP-Hello-ReqPDU after power on.

Bit 1 – 15: DeviceInitiativeValue.Reserved
This field shall be set according to 3.4.2.2.

4.3.1.4.15 Coding of the field SignalValue

This field shall be coded as data type Unsigned16 according to Table 104.

Table 104 – SignalValue

Value (hexadecimal)	Meaning	Usage
0x0100	Flash Once	Flash an LED (for example the Ethernet LINK LED) or an alternative signaling with duration of 3 s with a frequency of, for example, 1 Hz (500 ms on, 500 ms off).

4.3.1.4.16 Coding of the field NameOfStationValue

4.3.1.4.16.1 General

Each interface of a device shall be equipped with a unique NameOfStation. Thus, this field could be seen as NameOfInterface.

Historically most devices owned only one interface, thus the name assigned to the interface was used as NameOfStation.

The default value is an empty string. This indicates that no NameOfStation / NameOfInterface is assigned.

4.3.1.4.16.2 Encoding

This field shall be coded as data type OctetString with 1 to 240 octets following the definition of IETF RFC 5890 and the following syntax:

– 1 or more labels, separated by [.]
– Label length is 1 to 63 octets
– Total length is 1 to 240 octets
– Labels consist of [a-z0-9-]
– Labels do not start with [-]
– Labels do not end with [-]

Additional for NameOfStation / NameOfInterface:

– The first label does not have the form “port-xyz” or “port-xyz-abcde” with a, b, c, d, e, x, y, z = 0…9, to avoid wrong similarity with the field AliasNameValue
– Station-names do not have the form a.b.c.d with a, b, c, d = 0…999

NOTE 1 Labels do only start with ‘xn--‘ if the original string contains characters other than [a-z0-9-].

EXAMPLE 1 “device-1.machine-1.plant-1.vendor”

EXAMPLE 2 “mühle1.ölmühle1.plant.com“ is coded as “xn--mhle1-kva.xn--lmhle1-vxa4c.plant.com“

NOTE 2 This field is not terminated by zero.

NOTE 3 Avoiding the use of multiple concatenated [-] in labels except for ‘xn--‘simplifies the checking.

4.3.1.4.16.3 Semantics with Identify service

The following rules shall be applied:

– Name of station with length 1 to 240 octets:
Only devices with the same name shall answer. In the comparison of the two names for equality the DNS conventions shall be considered.

– Name of station with length == 0:
Only those devices shall answer, which have not received a station name yet.

The network representation of the NameOfStation shall be according to 4.3.1.4.16.2.

4.3.1.4.16.4 Semantics with Set service

The following rules shall be applied:

– Name of station with length 1 to 240 octets:
The station shall set its name according to the NameOfStationValue.

– Name of station with length == 0:
The station shall delete its stored name.

The network representation of the NameOfStation shall be according to 4.3.1.4.16.2.

4.3.1.4.17 Coding of the field NameOfPort

This field shall be coded as OctetString[8] or OctetString[14] as “port-xyz” or “port-xyz-rstuv” where x, y, z is in the range “0”-“9” from 001 up to 255 and r, s, t, u, v is in the range “0”-“9” from 00000 up to 32767. The values x, y, z shall be used to identify the port number. The values r, s, t, u, v shall be used to identify the slot which contains the port.

The value “port-001-00000” shall be used for the first port submodule of an interface within slot 0 if any other slot may contain another port submodule.

The value “port-001” shall be used for the first port submodule of an interface if no other slot can contain another port submodule.

Furthermore, the definition of IETF RFC 5890 shall be applied.

4.3.1.4.18 Coding of the field AliasNameValue

4.3.1.4.18.1 Encoding

This field shall be coded as OctetString. The content shall be the concatenation of the contents of the fields NameOfPort and NameOfStation.

$$\text{AliasNameValue} = \text{NameOfPort} + \text{“.”} + \text{NameOfStation} \quad (10)$$

where

AliasNameValue	is an additional name of a node. It addresses one particular interface of this node.
NameOfPort	is the neighbor’s name of port
NameOfStation	is the neighbor’s name of station

4.3.1.4.18.2 Semantics with Identify service

The following rules shall be applied:

– Alias name with length != 0:
Only the device shall answer with exactly the same additional alias name. In the comparison of the two names for equality the DNS conventions shall be considered. The device shall not answer with the AliasName. Instead, it shall use the current NameOfStation (even if the NameOfStation is not set).

– Alias name with length == 0:
Not allowed in a request.

NOTE The AliasName is a unique alternative name for a device or an interface of a device. The alias name is derived from topology or neighborhood information. The alias name of station cannot be set or gotten through means of DCP.

4.3.1.4.19 Coding of the field DeviceRoleDetails

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0: DeviceRoleDetails.IOdevice

This field shall be coded with the values according to Table 105.

Table 105 – DeviceRoleDetails.IO Device

Value (hexadecimal)	Meaning
0x00	No IO device instance.
0x01	The device/interface contains an IO device instance.

Bit 1: DeviceRoleDetails.IOcontroller

This field shall be coded with the values according to Table 106.

Table 106 – DeviceRoleDetails.IOcontroller

Value (hexadecimal)	Meaning
0x00	No IO controller instance.
0x01	The device/interface contains an IO controller instance.

Bit 2: DeviceRoleDetails.IOMultiDevice

This field shall be coded with the values according to Table 107.

Table 107 – DeviceRoleDetails.IOMultiDevice

Value (hexadecimal)	Meaning
0x00	No or one IO device instance.
0x01	The device/interface contains multiple IO device instances which can be active concurrently.

Bit 3: DeviceRoleDetails.IOsupervisor

This field shall be coded with the values according to Table 108.

Table 108 – DeviceRoleDetails.IOsupervisor

Value (hexadecimal)	Meaning
0x00	No IO supervisor instance.
0x01	The device/interface contains an IO supervisor instance.

Bit 4 – 7: DeviceRoleDetails.Reserved

This field shall be set to zero.

4.3.1.4.20 Coding of the field MACAddressValue

This field shall be coded as data type OctetString[6]. The value of the field shall be according to the 48-bit universal LAN MAC addresses of IEEE Std 802.

4.3.1.4.21 Coding section related to IPsuite

4.3.1.4.21.1 General

The rules for the fields in 4.3.1.4.21 differ between one interface and multiple (more than one) interface devices.

For multiple interface devices additional rules from the standards covering IP routing apply.

4.3.1.4.21.2 Coding of the field IPAddress

4.3.1.4.21.2.1 General

This field shall be coded as data type OctetString[4]. The encoding of the fields shall be according to IETF RFC 791, IETF RFC 6890 and Table 109.

The value zero in all octets shall be used to delete a currently set IP address.

Table 109 – IPAddress

Value (CIDR [a])	Comment	Range	Meaning
0.0.0.0 /32	Mandatory	0.0.0.0 up to 0.0.0.0	Special case: No IPsuite assigned in conjunction with SubnetMask and StandardGateway set to zero
0.0.0.0 /8	Invalid	0.0.0.0 up to 0.255.255.255	Reserved according to IETF RFC 6890
127.0.0.0 /8	Invalid	127.0.0.0 up to 127.255.255.255	Reserved according to IETF RFC 6890; loopback address
224.0.0.0 /4	Invalid	224.0.0.0 up to 239.255.255.255	Reserved according to IETF RFC 6890; IPv4 multicast address assignments
240.0.0.0 /4	Invalid	240.0.0.0 up to 255.255.255.255	Reserved according to IETF RFC 6890; reserved for future addressing modes and limited broadcast address (255.255.255.255)
Reserved	Invalid	—	Subnet part of the IPAddress is “0”
Reserved	Invalid	—	Host part of the IPAddress is a series of consecutive “1” (subnet broadcast address) IPAddress may be accepted but should be invalid.

Value (CIDR [a])	Comment	Range	Meaning
Reserved	Invalid	—	Host part of the IPAddress is a series of consecutive “0” (subnet address) IPAddress may be accepted but should be invalid.
Other	Mandatory	—	IP address assigned
[a] Notation according to classless inter domain routing defined in IETF RFC 4632			

NOTE The object IP suite (IPAddress, Subnetmask and StandardGateway) exists only once. The flag “store permanently” is an attribute of the object and changed in conjunction with every write access.

4.3.1.4.21.2.2 Single interface device

Check defined in Table 109 applies.

4.3.1.4.21.2.3 Multiple interface device

Check defined in Table 109 applies per interface.

However, the IP stack within the device shall have an unambiguous routing decision for outgoing packets.

If the device supports IETF RFC 6918, the routing Table will sort routing entries in reverse order according to the length of their subnet mask (for example first with subnet mask /32 (if applicable), then /31, and so on). The routing decision is in this case unambiguous, as long as each of the device’s interfaces has a unique subnet-number; note that the subnets can overlap in this case.

Otherwise, the subnets of the interfaces shall not overlap, for example for two interfaces x and y in each case the equation

(IFx_IPAddress & IFx_SubnetMask) != ((IFy_IPAddress & IFy_SubnetMask) & IFx_SubnetMask)

applies.

In case the IPAddress range is ambiguous, then the diagnosis “Multiple interface mismatch / IPAddress range of the interface is not unambiguous” appears on the related interfaces of the device.

EXAMPLE Consider IF1 with IPAddress := 192.168.215.10 and SubnetMask := 255.255.0.0, IF2 with IPAddress := 192.168.216.10 and SubnetMask := 255.255.255.0. In this case an outgoing packet having an IP address in range 192.168.216.0 to 192.168.216.255 is sent via IF2; having an IP address in range 192.168.0.0 to 192.168.215.255 resp. 192.168.217.0 to 192.168.255.255 via IF1.

4.3.1.4.21.3 Coding of the field Subnetmask

This field shall be coded as data type OctetString[4]. The encoding of the fields shall be according to IETF RFC 791, IETF RFC 6890 and to Table 110.

Table 110 – Subnetmask

Value	CIDR suffix	Number of hosts	Meaning
255.255.255.255	/ 32	1	Optional
255.255.255.254	/ 31	2	
255.255.255.252	/ 30	4	Subnet mask assigned
255.255.255.248	/ 29	8	
255.255.255.240	/ 28	16	
255.255.255.224	/ 27	32	
255.255.255.192	/ 26	64	
255.255.255.128	/ 25	128	
255.255.255.000	/ 24	256	
255.255.254.000	/ 23	512	
255.255.252.000	/ 22	1 024	
255.255.248.000	/ 21	2 048	
255.255.240.000	/ 20	4 096	
255.255.224.000	/ 19	8 192	
255.255.192.000	/ 18	16 384	
255.255.128.000	/ 17	32 768	
255.255.000.000	/ 16	65 536	
255.254.000.000	/ 15	131 072	
255.252.000.000	/ 14	262 144	
255.248.000.000	/ 13	524 288	
255.240.000.000	/ 12	1 048 576	
255.224.000.000	/ 11	2 097 152	
255.192.000.000	/ 10	4 194 304	
255.128.000.000	/ 9	8 388 608	
255.000.000.000	/ 8	16 777 216	
254.000.000.000	/ 7	33 554 432	Optional
252.000.000.000	/ 6	67 108 864	
248.000.000.000	/ 5	134 217 728	
240.000.000.000	/ 4	268 435 456	
224.000.000.000	/ 3	536 870 912	
192.000.000.000	/ 2	1 073 741 824	
128.000.000.000	/ 1	2 147 483 648	
000.000.000.000	/ 0	4 294 967 296	
0.0.0.0	—	—	Default value Used for “no IP-suite assigned”
Other	—	Reserved	Invalid subnet mask

4.3.1.4.21.4 Coding of the field StandardGateway

4.3.1.4.21.4.1 General

This field shall be coded as data type OctetString[4]. The encoding of the field shall be according to IETF RFC 791, IETF RFC 6890, Table 109 and Table 111.

If no IPAddress is assigned, the values of IPAddress, Subnetmask and StandardGateway shall be set to zero.

Table 111 – StandardGateway

Value (CIDR [a])	Meaning
0.0.0.0 / 32	Default value No standard gateway assigned The application shall set up the local IP implementation to avoid IP routing. If this interface supports routing, then the router shall be disabled.
Own IPAddress / 32 [b]	Node with routing support: – Standard gateway assigned – If this node supports routing, then the router shall be enabled/used. Node without routing support: – No standard gateway assigned – The application shall set up the local IP implementation to avoid IP routing.
Other [c]	Standard gateway assigned Should be checked against the field subnetmask
Reserved [d]	No standard gateway assigned
[a] Notation according to classless inter domain routing defined in IETF RFC 4632. [b] Should only be used for nodes with routing support. For nodes without routing support only 0.0.0.0 should be used. [c] Valid according to Table 109 and Table 110. [d] Invalid according to Table 109 and Table 110.	

4.3.1.4.21.4.2 Single interface device

Check defined in Table 111 applies.

An assigned StandardGateway shall be IP-reachable in the context of this interface. Thus, IPAddress and StandardGateway need to be accessible in the IP subnet defined by Subnetmask.

4.3.1.4.21.4.3 Multiple interface device

Check defined in Table 111 applies per interface.

An assigned StandardGateway shall be IP-reachable in the context of at least one available interface. Thus, IPAddress and StandardGateway need to be accessible in the IP subnet defined by Subnetmask for at least one available interface.

Individual StandardGateways may be assigned per interface of the device. However, the IP stack within the device has one StandardGateway for all interfaces. This also includes those interfaces which are not configured via this specification, for example IP only interfaces.

The following rules apply for the effective StandardGateway of the device:

– If no StandardGateway is assigned, there is no effective gateway. Any existing diagnosis information related to the gateway shall be cleared.

  No means that no interface has an assigned StandardGateway.

– If exactly one StandardGateway is assigned, then this is the effective gateway and applied to the IP stack. If a “StandardGateway mismatch” diagnosis was indicated before, then it will be cleared.

  Exactly one means that at least one interface shall have an assigned StandardGateway and all interfaces with assigned StandardGateway have the same gateway address.

– If several different gateways have been assigned, diagnosis “Multiple interface mismatch / standard gateway mismatch” appears on all interfaces of the device, indicating via ExtChannelAddValue the StandardGateway currently effective. Whether an effective StandardGateway is elected in this case and which one was chosen is a local matter.

  Several different means that for at least two interfaces StandardGateway is assigned with different gateway addresses.

– If the effective StandardGateway is not reachable from the device using this or some other interface, the diagnosis “Multiple interface mismatch / standard gateway inactive” appears on all interfaces, indicating which StandardGateway is currently inactive.

4.3.1.4.21.5 Correlations between the subfields of IPsuite

Table 112 shows the correlation between Table 109, Table 110, and Table 111.

Table 112 – Correlation between the subfields of IPsuite

Table 109 value	Table 110 value	Table 111 value	Meaning
0.0.0.0	0.0.0.0	0.0.0.0	Default value No IPsuite assigned or IPsuite reset
Valid	Valid	0.0.0.0	Mandatory IPAddress and Subnetmask assigned; no StandardGateway
Valid [a]	Valid	Valid [a]	Mandatory IPAddress and Subnetmask assigned; no StandardGateway
Valid [b]	Valid	Valid [b]	Mandatory IPAddress, Subnetmask, and StandardGateway assigned
Invalid	—	—	IPsuite rejected
—	Invalid	—	IPsuite rejected
—	—	Invalid	IPsuite rejected
[a] Value identical [b] Value different			

4.3.1.4.22 Coding of the fields DHCPParameterValue using DHCP Option 61

4.3.1.4.22.1 General

The client identifier can be used by a DHCP client to request its IP parameters from a DHCP server. A device shall be able to use the client identifier from the data set in the following way.

4.3.1.4.22.2 Coding of the field DHCPParameterLength

This field shall be coded as data type Unsigned8 and shall contain the length of the specific subsequent data.

When a supported Suboption of the Option DHCP doesn’t contain any data, for example is not set, the DHCPParameterLength:=0 shall be used.

4.3.1.4.22.3 Use of MACAddress as client identifier

If the device shall obtain its IP parameters using the DHCP protocol and shall use the MAC address as a client identifier, then option 61 shall have the values according to Table 113.

Table 113 – MACAddress as client identifier

Parameter name	Parameter value (decimal)
SuboptionDHCP	61
DHCPParameterLength	1
DHCPParameterData	1

4.3.1.4.22.4 Use of NameOfStation as client identifier

If the device shall obtain its IP parameters using the DHCP protocol and shall use the NameOfStation as client identifier, then option 61 shall have the values according to Table 114 (in addition to IETF RFC 2132).

Table 114 – NameOfStation as client identifier

Parameter name	Parameter value (decimal)
SuboptionDHCP	61
DHCPParameterLength	1
DHCPParameterData	0

If the name of station of the device has been changed the DHCP request shall automatically use the new NameOfStation in the next DHCP request.

4.3.1.4.22.5 Use of arbitrary client identifier

If the device shall obtain its IP parameters using the DHCP protocol and shall use an arbitrary string as client identifier, then option 61 shall have the values according to Table 115.

Table 115 – Arbitrary client identifier

Parameter name	Parameter value (decimal)
SuboptionDHCP	61
DHCPParameterLength	1 + length of DHCPParameterData[1…n]
DHCPParameterData[0]	0
DHCPParameterData[1…n]	An OctetString, at least two octets, uniquely identifying the interface, following the coding rules of the client identifier.

4.3.1.4.23 Coding of the fields DHCPParameterValue using DHCP Option 255

The DHCP option 255 is defined as END by IETF RFC 2131. It is used automatically by the DHCP client or server. Thus, this option is overridden by DCP and shall be used according to Table 116.

Table 116 – DHCPParameterValue using DHCP Option 255

Parameter name	Parameter value (decimal)
SuboptionDHCP	255
DHCPParameterLength	1
DHCPParameterData	0: Do not use DHCP 1: Legacy Do not use DHCP but set all DHCPOptions to their “Reset to factory” value 2: Default Use DHCP with the given set of DHCPOptions

NOTE If DHCP is enabled, then all earlier set IPAddress, Subnetmask and StandardGateway values are deleted. These fields are now controlled by DHCP.

This option should be the last one in a list of DHCP options in a set request.

4.3.1.4.24 Coding of the field ManufacturerOUI

This field shall be coded as OctetString[3] with the Organizationally Unique Identifier (OUI) as defined by IEEE Std 802.

NOTE The value of this central administrative number is given by the IEEE Registration Authority Committee. Available at <standards.ieee.org/regauth>

4.3.1.4.25 Coding of the field ManufacturerSpecificString

This field shall be coded as data type OctetString and shall contain the values of the related option and suboptions. If the BlockValue has an odd length a padding octet shall be added at the end in order to be Unsigned16 aligned. The padding octet shall have the value 0.

4.3.1.4.26 Coding of the field DeviceVendorValue

This field shall be coded as data type VisibleString and shall contain the same content as the field DeviceType without the trailing blanks.

4.3.1.4.27 Coding of the field DHCPParameterData

This field shall be coded as data type OctetString and shall contain the values of the related option and suboptions. If the BlockValue has an odd length a padding octet shall be added at the end in order to be Unsigned16 aligned. The padding octet shall have the value 0.

4.3.1.4.28 Coding of the field AddDataValue

This field shall be coded as data type OctetString and shall contain the values of the related option and suboptions. If the BlockValue has an odd length a padding octet shall be added at the end in order to be Unsigned16 aligned. The padding octet shall have the value 0.

4.3.1.4.29 Coding of the field StandardGatewayValue

The coding of this field shall be according to 3.4.2.3.4. and the individual bits shall have the following meaning:

Bit 0: StandardGatewayValue.StandardGateway
This field shall be coded with the values according to Table 117.

Table 117 – StandardGatewayValue.StandardGateway

Value (hexadecimal)	Meaning	Use
0x00	OwnIP is treated as “no gateway”	—
0x01	Standard gateway supported	This device / interface supports a standard gateway for routing.

Bit 1 – 15: StandardGatewayValue.Reserved
This field shall be coded with zero.

4.3.1.4.30 Coding of the field RsiPropertiesValue

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall be coded with the values according to Table 118.

Table 118 – RsiPropertiesValue

Bit	Value	Meaning
0	0	IP stack not available
	1	IP stack available
1	0	CLRPC Interface not available
	1	CLRPC Interface available
2	0	RSI AR Interface not available
	1	RSI AR Interface available
3	0	RSI AR Read Implicit Interface not available
	1	RSI AR Read Implicit Interface available
4	0	RSI CIM Interface not available
	1	RSI CIM Interface available
5	0	RSI CIM Read Implicit Interface not available
	1	RSI CIM Read Implicit Interface available
6 – 15	Shall be set according to 3.4.2.2	

4.3.1.4.31 Coding of the field NMEPrio

This field shall be coded as data type Unsigned16 and shall be set according to Table 119.

Table 119 – NMEPrio

Value (hexadecimal)	Meaning
0x0000	Highest priority NME
0x0001 – 0x3000	High priorities for NME
0x3001 – 0x9FFF	Low priorities for NME
0xA000	Lowest priority for NME Default priority for NME
0xA001 – 0xFFFF	Reserved

4.3.2 DCP protocol state machines

4.3.2.1 FAL Service Protocol Machines

There is no FAL Service Protocol Machine (FSPM) defined for this protocol.

4.3.2.2 Application Relationship Protocol Machines

4.3.2.2.1 DCP Unicast Sender

4.3.2.2.1.1 Primitive definitions

####### 4.3.2.2.1.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DCPUCS are described in the service definition and shown in Table 120.

Table 120 – Remote primitives issued or received by DCPUCS

Primitive	Source	Destination	Associated parameters	Functions
DCP_Get.req	local means	DCPUCS	CREP, DA, ListOfOptions	This service is used to fetch data from the remote DCP ASE. The parameter DA contains the Layer2 address of the server and the parameter ListOfOptions the requested information.
DCP_Set.req	CTLDINA	DCPUCS	CREP, DA, ListOfData, ListOfControlCommands	This service is used to convey data to the remote DCP ASE. The parameter DA contains the Layer2 address of the server and the parameter ListOfData the requested information. The ListOfControlCommand contains steering information.
LMPM_A_Data.cnf (-)	LMPM	DCPUCS	CREP, LMPM_status	—
LMPM_A_Data.cnf (+)	LMPM	DCPUCS	CREP, LMPM_status	—
LMPM_A_Data.ind	LMPM	DCPUCS	CREP, DA, SA, Prio, A_SDU	—
DCP_Get.cnf (-)	DCPUCS	local means	ERRCLS, ERRCODE	—
DCP_Get.cnf (+)	DCPUCS	local means	ListOfData	—
DCP_Set.cnf (-)	DCPUCS	CTLDINA	ERRCLS, ERRCODE	—
DCP_Set.cnf (+)	DCPUCS	CTLDINA	CREP, ListOfResponse	This service conveys the data and the execution info.
LMPM_A_Data.req	DCPUCS	LMPM	CREP, DA, SA, Prio, A_SDU	—

4.3.2.2.1.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DCPUCS are described in the service definition and shown in Table 121.

Table 121 – Local primitives issued or received by DCPUCS

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.3.2.2.1.2 State transition diagram

The state transition diagram of the DCPUCS is shown in Figure 29.

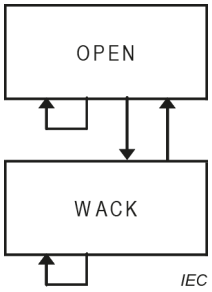


Figure 29 – State transition diagram of DCPUCS

States of the DCPUCS are:

OPEN Wait for a DCP_Get or DCP_Set request from the application and send it to the server. Then change to state WACK and wait for the response.

WACK Wait for a DCP_Get or DCP_Set response from the server and create a DCP_Get or DCP_Set confirmation.
If the server does not respond, retry the request and after the retry limit has expired, create a negative confirmation.

4.3.2.2.1.3 State machine description

The state machine gets its initial values from the DCP ASE attributes. During the OPEN state, DCP Get and Set services requests issued by the DCP user are being transformed to Data Link Layer services. After issuing the transmission of data, the WACK state is entered to wait for the Response-PDU causing a confirmation primitive to the DCP user.

4.3.2.2.1.4 DCPUCS state table

Table 122 contains the complete description of the DCPUCS state table.

Table 122 – DCPUCS state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**DCP_Get.req ()** => Pending:= GET DA := From instance identified by CREP XID := SXID A_SDU := DCP-Get-ReqPDU with parameters from the request Store A_SDU Retry:= Max Retry Limit StartTimer (UC Client Timeout) LMPM_A_Data.req ()	WACK
2	OPEN	**DCP_Set.req ()** => Pending:= SET DA := From instance identified by CREP XID := SXID A_SDU := DCP-Set-ReqPDU with parameters from the request Store A_SDU Retry:= Max Retry Limit StartTimer (UC Client Timeout) LMPM_A_Data.req ()	WACK
3	OPEN	**LMPM_A_Data.ind ()** => ignore	OPEN

#	Current State	Event /Condition =>Action	Next State
4	OPEN	**LMPM_A_Data.cnf ()** => ignore	OPEN
5	OPEN	**TimerExpired (UC Client Timeout)** => ignore	OPEN
6	WACK	**DCP_Get.req ()** => ERRCLS := CTXT ERRCODE := INVALID_STATE DCP_Get.cnf (-)	WACK
7	WACK	**DCP_Set.req ()** => ERRCLS := CTXT ERRCODE := INVALID_STATE DCP_Set.cnf (-)	WACK
8	WACK	**LMPM_A_Data.ind ()** /DCP-Get-ResPDU && Pending == GET && XID == SXID => SXID := SXID+1 StopTimer (UC Client Timeout) DCP_Get.cnf (+)	OPEN
9	WACK	**LMPM_A_Data.ind ()** /DCP-Get-ResPDU && (Pending != GET || XID != SXID) => ignore	WACK
10	WACK	**LMPM_A_Data.ind ()** /DCP-Set-ResPDU && Pending == SET && XID == SXID => SXID:= SXID+1 StopTimer (UC Client Timeout) DCP_Set.cnf (+)	OPEN
11	WACK	**LMPM_A_Data.ind ()** /DCP-Set-ResPDU && (Pending != SET || XID != SXID) => ignore	WACK
12	WACK	**LMPM_A_Data.cnf (+)** /LMPM_status == OK => ignore	WACK
13	WACK	**LMPM_A_Data.cnf (-)** /LMPM_status != OK && Pending == GET => ERRCLS := PROTOCOL ERRCODE := LMPM StopTimer (UC Client Timeout) DCP_Get.cnf (-)	OPEN

#	Current State	Event /Condition =>Action	Next State
14	WACK	**LMPM_A_Data.cnf (-)** /LMPM_status != OK && Pending == SET => ERRCLS := PROTOCOL ERRCODE := LMPM StopTimer (UC Client Timeout) DCP_Set.cnf (-)	OPEN
15	WACK	**TimerExpired (UC Client Timeout)** /Retry != 0 => A_SDU := from Stored A_SDU Retry:= Retry-1 StartTimer (UC Client Timeout) LMPM_A_Data.req ()	WACK
16	WACK	**TimerExpired (UC Client Timeout)** /Retry == 0 && Pending == GET => ERRCLS := PROTOCOL ERRCODE := TIMEOUT DCP_Get.cnf (-)	OPEN
17	WACK	**TimerExpired (UC Client Timeout)** /Retry == 0 && Pending == SET => ERRCLS := PROTOCOL ERRCODE := TIMEOUT DCP_Set.cnf (-)	OPEN
18	WACK	**LMPM_A_Data.ind ()** /!(DCP-Set-ResPDU || DCP-Get-ResPDU) => ignore	WACK

4.3.2.2.1.5 Functions, Macros, Timers and Variables

Table 123 contains the functions, macros, timers and variables used by the DCPUCS and their arguments and their descriptions.

Table 123 – Functions, Macros, Timers and Variables used by the DCPUCS

Name	Type	Function/Meaning
StartTimer	Function	This function is used to start or restart a timer.
StopTimer	Function	This function is used to stop a timer.
TimerExpired	Function	This function signals that a timer has expired.
UC Client Timer	Timer	This timer, restarted on the requestor when sending an unicast request, will terminate waiting for a response.
ERRCLS	Variable	This local variable contains the entity signaling the error.
ERRCODE	Variable	This local variable contains the error reason in the context of the ERRCLS.
Max Retry Limit	Variable	This local variable contains the retry limit.
Pending	Variable	This local variable holds the information whether a Set or Get service is pending.
Retry	Variable	This local variable contains the retry counter. The maximum value shall be taken from the appropriate DCP ASE attribute.
SXID	Variable	This local variable contains the Transaction ID XID used to identify services uniquely. The initial value shall be derived from a random number generator.
UC Client Timeout	Variable	The value of this local variable shall be taken from the appropriate DCP ASE attribute. The default value is 1 s.

4.3.2.2.2 DCP Unicast Receiver

4.3.2.2.2.1 Primitive definitions

4.3.2.2.2.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DCPUCR are described in the service definition and shown in Table 124.

Table 124 – Remote primitives issued or received by DCPUCR

Primitive	Source	Destination	Associated parameters	Functions
LMPM_A_Data.cnf	LMPM	DCPUCR	CREP, LMPM_status	—
LMPM_A_Data.ind	LMPM	DCPUCR	CREP, DA, SA, Prio, A_SDU	—
LMPM_A_Data.req	DCPUCR	LMPM	CREP, DA, SA, Prio, A_SDU	—

4.3.2.2.2.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DCPUCR are described in the service definition and shown in Table 125.

Table 125 – Local primitives issued or received by DCPUCR

Primitive	Source	Destination	Associated parameters	Functions
CIM_DatabaseUpdate_ind	CIM	*	—	—

4.3.2.2.2.2 State transition diagram

The state transition diagram of the DCPUCR is shown in Figure 30.

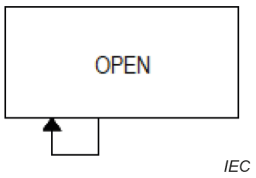


Figure 30 – State transition diagram of DCPUCR

States of the DCPUCR are:

OPEN Wait for a DCP Get or DCP Set service. Prepare the response for GET from CIM database and store SET in CIM database.

Changing the content of the CIM database triggers additonal actions.

4.3.2.2.2.3 State machine description

The state machine is waiting for LMPM_A_Data service indication. After pARsing the indication, an appropriate action is taken. The state machine responds with a LMPM_A_Data service request containing the data or status of the indicated DCP service.

Updating the CIM database creates additional action, for example starting or stopping the IP-stack, starting or stopping DHCP.

4.3.2.2.2.4 DCPUCR state table

Table 126 contains the complete description of the DCPUCR state table. A LMPM_A_Data.ind primitive will be accepted if its type is DCP-Get-ReqPDU or DCP-Set-ReqPDU.

Table 126 – DCPUCR state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**LMPM_A_Data.ind ()** /DCP-Get-ReqPDU && (SAM == NIL || SAM == SA) && (DA != Multicast) && CheckAPDU () == Valid => SAM := SA SXID := XID SDA := SA DA := SDA XID := SXID A_SDU := CreateDCP-Get-RspPDU (ListOfData) StartTimer (Client Hold Time) LMPM_A_Data.req ()	OPEN

#	Current State	Event /Condition =>Action	Next State
2	OPEN	**LMPM_A_Data.ind ()** /DCP-Get-ReqPDU && (SAM == NIL || SAM == SA) && (DA != Multicast) && (CheckAPDU () == Valid, ProtectionOn || CheckAPDU () == Invalid) => SAM := SA SXID := XID SDA := SA DA := SDA XID := SXID ReturnCodeFromCheckAPDU A_SDU := CreateDCP-Get-RspPDU with status/error information StartTimer (Client Hold Time) LMPM_A_Data.req ()	OPEN
3	OPEN	**LMPM_A_Data.ind ()** /DCP-Set-ReqPDU && (SAM == NIL || SAM == SA) && (DA != Multicast) && CheckAPDU () == Invalid, RspTooLong => ignore	OPEN
4	OPEN	**LMPM_A_Data.ind ()** /DCP-Set-ReqPDU && (SAM == NIL || SAM == SA) && (DA != Multicast) && (CheckAPDU () == Valid, ProtectionOn || CheckAPDU () == Valid, RejectDCPSet || CheckAPDU () == Valid, RejectARActive || CheckAPDU () == Invalid || CheckAPDU () == Valid, Unknown) => SAM := SA SXID := XID SDA := SA DA := SDA XID := SXID ReturnCodeFromCheckAPDU A_SDU := CreateDCP-Set-RspPDU with status/error information StartTimer (Client Hold Time) LMPM_A_Data.req ()	OPEN

#	Current State	Event /Condition =>Action	Next State
5	OPEN	**LMPM_A_Data.ind ()** /DCP-Set-ReqPDU && (SAM == NIL || SAM == SA) && (DA != Multicast) && (CheckAPDU () == Valid || CheckAPDU () == Valid, Name, IP || CheckAPDU () == Valid, Name, DHCP || CheckAPDU () == Valid, ResetIP || CheckAPDU () == Valid, ResetName || CheckAPDU () == Valid, ResetToFactory) => SAM := SA SXID := XID SDA := SA DA := SDA XID := SXID StoreDataInCIM () A_SDU := CreateDCP-Set-RspPDU (ListOfData) containing the response from CheckAPDU and CIM database StartTimer (Client Hold Time) CIM_DatabaseUpdate_ind () LMPM_A_Data.req ()	OPEN
6	OPEN	**LMPM_A_Data.ind ()** /DCP-Get-ReqPDU && (SAM != NIL && SAM != SA) => ignore	OPEN
7	OPEN	**LMPM_A_Data.ind ()** /DCP-Set-ReqPDU && (SAM != NIL && SAM != SA) => ignore	OPEN
8	OPEN	**LMPM_A_Data.ind ()** /! DCP-Set-ReqPDU && ! DCP-Get-ReqPDU => ignore	OPEN
9	OPEN	**LMPM_A_Data.cnf ()** => ignore	OPEN
10	OPEN	**TimerExpired (Client Hold Time)** => SAM := NIL	OPEN

4.3.2.2.2.5 Functions, Macros, Timers and Variables

Table 127 contains the functions, macros, timers and variables used by the DCPUCR and their arguments and their descriptions.

Table 127 – Functions, Macros, Timers and Variables used by the DCPUCR

Name	Type	Function/Meaning
CreateDCP-Get-RspPDU	Function	Create DCP-Get-ResPDU with ListOfData taken from the CIM database according CheckAPDU from the DCP-Get-ReqPDU.
CreateDCP-Set-RspPDU	Function	Create DCP-Set-ResPDU with ListOfData taken from the CIM database according CheckAPDU from the DCP-Set-ReqPDU.
StartTimer	Function	This local function is used to start or restart a timer.
StopTimer	Function	This local function is used to stop a timer.
StoreDataInCIM	Function	This function stores the data from DCPSet, according to the feedback from CheckAPDU, in the CIM database
TimerExpired	Function	This local function is issued if a timer expires.
DCP-Get-ReqPDU	Macro	This macro checks the LMPM_A_Data.ind against the DCP APDU abstract syntax. The content of the field value is not checked.
DCP-Set-ReqPDU	Macro	This macro checks the LMPM_A_Data.ind against the DCP APDU abstract syntax. The content of the field value is not checked.
ListOfData	Macro	If the requested CheckAPDU do not fit into a DCP-Get-ResPDU, then fill into ListOfData as long as the requested Options/Suboptions fit in completely. Any residues from CheckAPDU shall be skipped.
CheckAPDU	Macro	This macro checks the PDU and return meaning for the state machine. See Table 128.
ReturnCodeFromCheckAPDU	Macro	See Table 128.
Client Hold Timer	Timer	This timer is used to protect the server from multiple client access (in conjunction with SAM) during state OPEN. It protects the client reaction time for the next request and is independent from the server reaction time.
Client Hold Time	Variable	The default value of this time is 3 s.
SAM	Variable	This local variable contains the SA of the client which is at the moment "owner" of the server.
SDA	Variable	This local variable contains the last valid SA conveyed with a LMPM_A_Data indication with valid data and is used for the generating of a response.
SXID	Variable	This local variable contains the Transaction ID XID used to identify services uniquely in the context of the client. The value shall be taken from the request PDU.

Table 128 – Return values for CheckAPDU

Name	Function/Meaning
ChangeIP	ChangeIP states that the frame contains an IP-suite and a different IP-suite is already stored. No action for the DCP ASE if IP-suite is identical.
ChangeName	ChangeName states that the frame contains a NameOfStation and a different NameOfStation is already stored. No action for the DCP ASE if NameOfStation is identical. If an AR exists, abort reason "DCP – station-name changed" shall be used.
DHCP	DHCP states that the frame contains DHCP enable.
Invalid	Invalid states that the frame is incorrect according to this document.
IP	IP states that the frame contains an IP-suite.
Name	Name states that the frame contains the NameOfStation.
ProtectionOn	TRUE if access through unprotected DCPset and DCPget commands is prohibited, otherwise FALSE.

Name	Function/Meaning
RejectARactive	If an AR exists, reject DCPset (IPsuite).
RejectDCPset	If an AR with ARProperties.RejectDCPsetRequests:=Rejected exists, then reject all DCPset commands except ControlOption.SuboptionSignal.
ResetIP	ResetIP states that the frame contains a delete IP-suite command.
ResetName	ResetName states that the frame contains a delete NameOfStation command. If an AR exists, abort reason "DCP – station-name changed" shall be used.
ResetToFactory	ResetToFactory (or FactoryReset) states that the frame contains a reset command (includes delete NameOfStation). If an AR exists, abort reason "DCP – reset to factory or factory reset" shall be used.
RspTooLong	A correct DCP_Set response would be larger than the allowed frame size
Unknown	Unknown states that the frame contains unknown options or suboptions. If the nodes doesn't support an IP-Stack, then IP-suite or ResetIP may be handled as Unknown, too.
Valid	Valid states that the frame is correct according to this document. Valid allows additional return values to provide further details.

4.3.2.2.3 DCP Multicast Sender

4.3.2.2.3.1 Primitive definitions

4.3.2.2.3.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DCPMCS are described in the service definition and shown in Table 129.

Table 129 – Remote primitives issued or received by DCPMCS

Primitive	Source	Destination	Associated parameters	Functions
DCP_Identify.req	CTLDINA	DCPMCS	ListOfFilter, ResponseDelayFactor	—
LMPM_A_Data.cnf (-)	LMPM	DCPMCS	CREP, LMPM_status	—
LMPM_A_Data.cnf (+)	LMPM	DCPMCS	CREP, LMPM_status	—
LMPM_A_Data.ind	LMPM	DCPMCS	CREP, DA, SA, Prio, A_SDU	—
DCP_Identify.cnf (-)	DCPMCS	CTLDINA	CREP, ERRCLS, ERRCODE	—
DCP_Identify.cnf (+)	DCPMCS	CTLDINA	ListOfDevices	—
LMPM_A_Data.req	DCPMCS	LMPM	CREP, DA, SA, Prio, A_SDU	—

4.3.2.2.3.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DCPMCS are described in the service definition and shown in Table 130.

Table 130 – Local primitives issued or received by DCPMCS

Primitive	Source	Destination	Associated parameters	Functions
DCP_Identify_ind	DCPMCS	CTLDINA	SA, ListOfDevices	—

4.3.2.2.3.2 State transition diagram

The state transition diagram of the DCPMCS is shown in Figure 31.

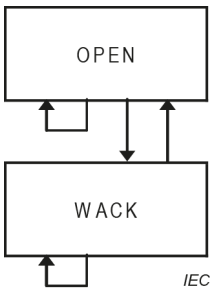


Figure 31 – State transition diagram of DCPMCS

States of the DCPMCS are:

OPEN Wait for an Identify service request.

WACK Wait for all responses for a defined time.

4.3.2.2.3.3 State machine description

The machine waits in the OPEN state for an Identify service request. After issuing the transmission of data the WACK state is entered to wait for all responses for a defined time. The timeout causes the issuing of the Identify confirmation service primitive with all received responses to the DCP user and returns the state machine to the OPEN state.

4.3.2.2.3.4 DCPMCS state table

Table 131 contains the complete description of the DCPMCS state machine. A LMPM_A_Data.ind primitive shall be accepted if its type is DCP-IdentifyResPDU.

Table 131 – DCPMCS state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**DCP_Identify.req ()** => XID := SXID DA := DCP_MulticastMACAdd A_SDU := DCP-Identify-ReqPDU First := TRUE StartTimer (Response delay timeout) LMPM_A_Data.req ()	WACK
2	OPEN	**LMPM_A_Data.ind ()** => ignore	OPEN

#	Current State	Event /Condition =>Action	Next State
3	OPEN	**LMPM_A_Data.cnf ()** => ignore	OPEN
4	OPEN	**TimerExpired (Response delay timeout)** => ignore	OPEN
5	WACK	**DCP_Identify.req ()** => ERRCLS := CTXT ERRCODE := INVALID_STATE DCP_Identify.cnf (-)	WACK
6	WACK	**LMPM_A_Data.ind ()** /(DCP-Identify-ResPDU && XID == SXID) && FIRST == FALSE => ListOfDevices.SA += SA ListOfDevices.ListOfData += A_SDU	WACK
7	WACK	**LMPM_A_Data.ind ()** /(DCP-Identify-ResPDU && XID == SXID) && First == TRUE => ListOfDevices.SA += SA ListOfDevices.ListOfData += A_SDU First := FALSE DCP_Identify_ind (SA, ListOfDevices)	WACK
8	WACK	**LMPM_A_Data.ind ()** /!(DCP-Identify-ResPDU && XID == SXID) => ignore	WACK
9	WACK	**LMPM_A_Data.cnf (+)** => ignore	WACK
10	WACK	**LMPM_A_Data.cnf (-)** => ERRCLS := PROTOCOL ERRCODE := LMPM StopTimer (Response delay timeout) DCP_Identify.cnf (-)	OPEN
11	WACK	**TimerExpired (Response delay timeout)** => ListOfDevices := Stored ListOfDevices SXID := SXID + 1 DCP_Identify.cnf (+)	OPEN

4.3.2.2.3.5 Functions, Macros, Timers and Variables

Table 132 contains the functions, macros, timers and variables used by the DCPMCS and their arguments and their descriptions.

Table 132 – Functions used by the DCPMCS

Name	Type	Function/Meaning
SXID	Variable	This local variable contains the Transaction ID XID used to identify services uniquely. It shall be changed with every request.
First	Variable	This variable is used for speeding up the connection establishment.
Response delay timer	Timer	This timer is used as timeout for the DCP Identify response.
Response delay timeout	Variable	The value is derived from the ResponseDelayFactor.
TimerExpired	Function	This function is issued if a timer expires.
StartTimer	Function	This function is used to start or restart a timer.
StopTimer	Function	This function is used to stop a timer.

4.3.2.2.4 DCP Multicast Receiver

4.3.2.2.4.1 General

Figure 32 shows the basic structure of the DCP Multicast Receiver (DCPMCR). The number of supported instances shall be at least one in each pool.

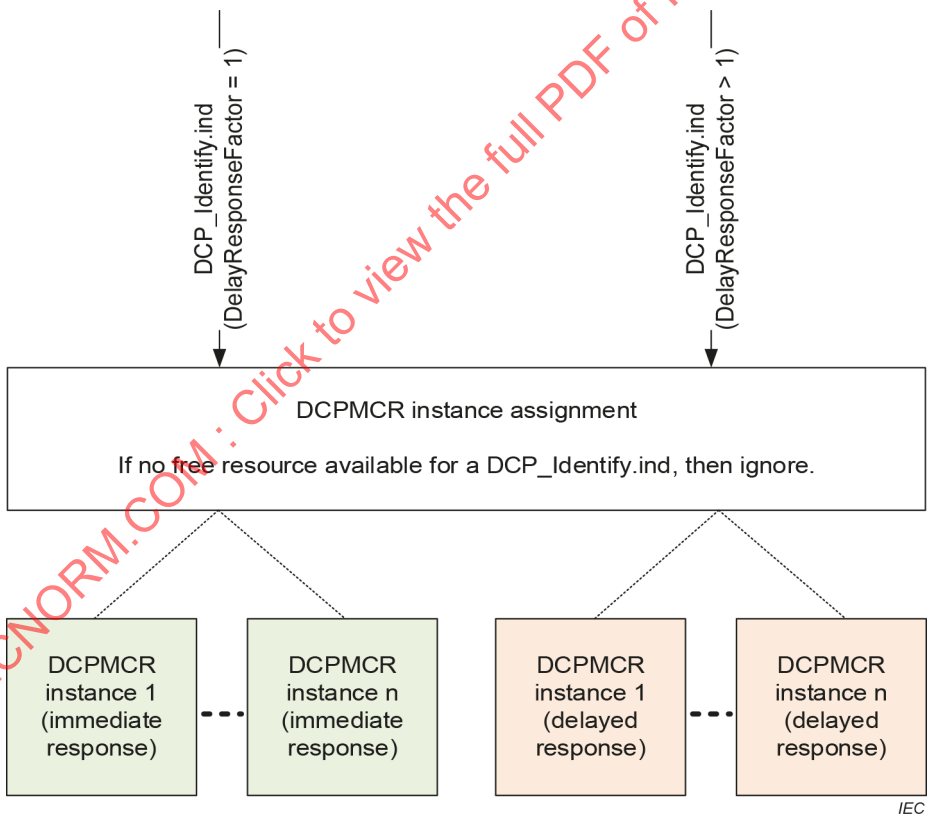


Figure 32 – Basic structure of a DCP Multicast Receiver

4.3.2.2.4.2 Primitive definitions

4.3.2.2.4.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DCPMCR are described in the service definition and shown in Table 133.

Table 133 – Remote primitives issued or received by DCPMCR

Primitive	Source	Destination	Associated parameters	Functions
LMPM_A_Data.cnf	LMPM	DCPMCR	CREP, LMPM_status	—
LMPM_A_Data.ind	LMPM	DCPMCR	CREP, DA, SA, Prio, A_SDU	—
LMPM_A_Data.req	DCPMCR	LMPM	CREP, DA, SA, Prio, A_SDU	—

4.3.2.2.4.2.2 Primitives exchanged between local machines

The local primitives including their associated parameters issued or received by DCPMCR are described in the service definition and shown in Table 134.

Table 134 – Local primitives issued or received by DCPMCR

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.3.2.2.4.3 State transition diagram

The state transition diagram of the DCPMCR is shown in Figure 33.

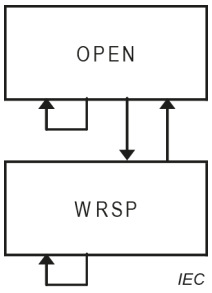


Figure 33 – State transition diagram of DCPMCR

States of the DCPMCR are:

OPEN Wait for DCP-IdentifyReqPDUs and compare the local data with the ListOfFilter. If a hit is detected start the response delay and change to the next state.

WRSP Wait for the response delay and send the response in case the DCP user has recognized a match with local data.

4.3.2.2.4.4 State machine description

The state machine waits for DCP-IdentifyReqPDUs in the OPEN state. The filter list shall be passed to the DCP user to check the filter criteria. The state machine waits in the state WRSP for a defined time which is calculated by means of the client's Response Delay Factor before sending the response in case the DCP user has recognized a match with local data. If no match is detected, nothing shall be sent.

4.3.2.2.4.5 DCPMCR state table

Table 135 contains the complete description of the DCPMCR state machine. A LMPM_A_Data.ind primitive shall be accepted if its type is DCP-IdentifyReqPDU.

Table 135 – DCPMCR state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**LMPM_A_Data.ind ()** /DCP-Identify-ReqPDU && Successful comparison with ListOfFilters => SAM := SA SXID := XID ResponseDelay := CalculateResponseDelay(ResponseDelayFactor, MACAddress) A_SDU := Create DCP-Identify-ResPDU with ListOfData taken from the DCP ASE according the ListOfFilters of the DCP-Identify-ReqPDU StartTimer (ResponseDelay)	WRSP
2	OPEN	**LMPM_A_Data.ind ()** /DCP-Identify-ReqPDU && ! Successful comparison with ListOfFilters => ignore	OPEN
3	OPEN	**LMPM_A_Data.ind ()** /!DCP-Identify-ReqPDU => ignore	OPEN
4	OPEN	**LMPM_A_Data.cnf ()** => ignore	OPEN
5	WRSP	**TimerExpired (ResponseDelay)** => DA := SAM XID := SXID LMPM_A_Data.req ()	OPEN
6	WRSP	**LMPM_A_Data.ind ()** => ignore	WRSP
7	WRSP	**LMPM_A_Data.cnf ()** => ignore	WRSP

4.3.2.2.4.6 Functions, Macros, Timers and Variables

Table 136 contains the functions, macros, timers and variables used by the DCPMCR and their arguments and their descriptions.

Table 136 – Functions, Macros, Timers and Variables used by the DCPMCR

Name	Type	Function/Meaning
StartTimer	Function	This local function starts or restarts a timer.
StopTimer	Function	This local function stops a timer.
TimerExpired	Function	This local function is issued if a timer expires.

Name	Type	Function/Meaning
CalculateResponseDelay	Macro	This local macro calculates the ResponseDelay according to the rules of this document using the ResponseDelayFactor and the interface MACAddress.
Successful comparison with ListOfFilters	Macro	Checks the PDU against the parameters stored in the DCP ASE.
ResponseDelayTimer	Timer	This timer is used to create the response delay. If a 100 ms timebased timer is used, then the ResponseDelay, if not zero, shall be rounded up to the next 100 ms value.
ResponseDelay	Variable	This variable contains the value according the calculation rules stated in this document.
SAM	Variable	This local variable contains the SourceAddress of the requester used for the response.
SXID	Variable	This local variable contains the Transaction ID XID used to identify services uniquely.

4.3.2.2.5 DCP Hello Multicast Sender

4.3.2.2.5.1 Primitive definitions

4.3.2.2.5.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DCPHMCS are described in the service definition and shown in Table 137.

Table 137 – Remote primitives issued or received by DCPHMCS

Primitive	Source	Destination	Associated parameters	Functions
DCP_Hello.req	CMINA	DCPHMCS	ListOfData	—
LMPM_A_Data.cnf (-)	LMPM	DCPHMCS	CREP, LMPM_status	—
LMPM_A_Data.cnf (+)	LMPM	DCPHMCS	CREP, LMPM_status	—
LMPM_A_Data.ind	LMPM	DCPHMCS	CREP, DA, SA, Prio, A_SDU	—
DCP_Hello.cnf (-)	DCPHMCS	CMINA	ERRCLS, ERRCODE	—
DCP_Hello.cnf (+)	DCPHMCS	CMINA	—	—
LMPM_A_Data.req	DCPHMCS	LMPM	CREP, DA, SA, Prio, A_SDU	—

4.3.2.2.5.1.2 Primitives exchanged between local machines

The local primitives including their associated parameters issued or received by DCPHMCS are described in the service definition and shown in Table 138.

Table 138 – Local primitives issued or received by DCPHMCS

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.3.2.2.5.2 State transition diagram

The state transition diagram of the DCPHMCS is shown in Figure 34.

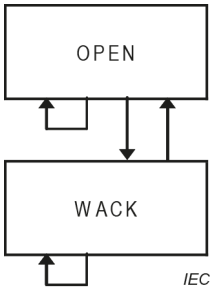


Figure 34 – State transition diagram of DCPHMCS

States of the DCPHMCS are:

OPEN Wait for a Hello service request from the upper layer.

WACK Wait for the Hello confirmation from the lower layer.

4.3.2.2.5.3 State machine description

The machine waits in the OPEN state for a Hello service request. After issuing the transmission of data the WACK state is entered to wait for the transmit confirmation.

4.3.2.2.5.4 DCPHMCS state table

Table 139 contains the complete description of the DCPHMCS state machine.

Table 139 – DCPHMCS state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**DCP_Hello.req ()** => DA := DCP_MulticastMACAdd for Hello A_SDU := DCP-Hello-ReqPDU LMPM_A_Data.req ()	WACK
2	OPEN	**LMPM_A_Data.ind ()** => ignore	OPEN
3	OPEN	**LMPM_A_Data.cnf ()** => ignore	OPEN
4	WACK	**LMPM_A_Data.ind ()** => ignore	WACK
5	WACK	**LMPM_A_Data.cnf (+)** => DCP_Hello.cnf (+)	OPEN
6	WACK	**LMPM_A_Data.cnf (-)** => ERRCLS := PROTOCOL ERRCODE := LMPM DCP_Hello.cnf (-)	OPEN

4.3.2.2.5.5 Functions, Macros, Timers and Variables

Table 140 contains the functions, macros, timers and variables used by the DCPHMCS and their arguments and their descriptions.

Table 140 – Functions, Macros, Timers and Variables used by the DCPHMCS

Name	Type	Function/Meaning
SXID	Variable	The XID shall be ignored by the Hello receiver. It may be incremented by the sender.

4.3.2.2.6 DCP Hello Multicast Receiver

4.3.2.2.6.1 Primitive definitions

4.3.2.2.6.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DCPHMCR are described in the service definition and shown in Table 141.

Table 141 – Remote primitives issued or received by DCPHMCR

Primitive	Source	Destination	Associated parameters	Functions
LMPM_A_Data.ind	LMPM	DCPHMCR	CREP, DA, SA, Prio, A_SDU	—
DCP_Hello.ind	DCPHMCR	CTLDINA	ListOfData	—

4.3.2.2.6.1.2 Primitives exchanged between local machines

The local primitives including their associated parameters issued or received by DCPHMCR are described in the service definition and shown in Table 142.

Table 142 – Local primitives issued or received by DCPHMCR

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.3.2.2.6.2 State transition diagram

The state transition diagram of the DCPHMCR is shown in Figure 35.

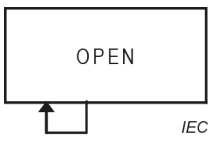


Figure 35 – State transition diagram of DCPHMCR

State of the DCPHMCR is:

OPEN Wait for DCP-Hello-ReqPDUs.

4.3.2.2.6.3 State machine description

The state machine waits for DCP-Hello-ReqPDUs in the OPEN state.

4.3.2.2.6.4 DCPHMCR state table

Table 143 contains the complete description of the DCPHMCR state machine. A LMPM_A_Data.ind primitive will be accepted if its type is DCP-Hello-ReqPDU.

Table 143 – DCPHMCR state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**LMPM_A_Data.ind ()** /DCP-Hello-ReqPDU && Successful comparison with ListOfData => DCP_Hello.ind ()	OPEN
2	OPEN	**LMPM_A_Data.ind ()** /!DCP-Hello-ReqPDU => ignore	OPEN

4.3.2.2.6.5 Functions, Macros, Timers and Variables

Table 144 contains the functions, macros, timers and variables used by the DCPHMCR and their arguments and their descriptions.

Table 144 – Functions, Macros, Timers and Variables used by the DCPHMCR

Name	Type	Function/Meaning
Successful comparison with ListOfData	Macro	Compare the PDU content with the local stored information and return TRUE if it fits.

4.3.3 DLL Mapping Protocol Machines

There is no DLL Mapping Protocol Machine (DMPM) defined for this Protocol.

4.4 Precision transparent clock protocol

4.4.1 FAL syntax description

4.4.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.4.1.2 APDU abstract syntax

Table 145 defines the abstract syntax of the PTCP PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs.

Table 145 – PTCP APDU syntax

APDU name	APDU structure
PTCP-PDU	PTCP-SYNC-PDU ^ PTCP-FUP-PDU ^ PTCP-ANNOUNCE-PDU ^ PTCP-DELAY-REQ-PDU ^ PTCP-DELAY-RSP-PDU ^ PTCP-DELAY-FU-RSP-PDU
PTCP-FUP-PDU	PTCP-Header-FUP, PTCP-FollowUpPDU
PTCP-SYNC-PDU	PTCP-Header-Sync, PTCP-RTSyncPDU
PTCP-ANNOUNCE-PDU	PTCP-Header-Announce, PTCP-AnnouncePDU
PTCP-DELAY-REQ-PDU	PTCP-Header-Delay, PTCP-DelayReqPDU
PTCP-DELAY-RSP-PDU	PTCP-Header-Delay, PTCP-DelayResPDU
PTCP-DELAY-FU-RSP-PDU	PTCP-Header-Delay, PTCP-DelayFuResPDU
PTCP-RTSyncPDU	PTCPSubdomain, PTCPTime, PTCPTimeExtension, PTCPMaster, [PTCPOption], PTCPEnd, [APDU_Status] [a]
PTCP-AnnouncePDU	PTCPSubdomain, PTCPMaster, [PTCPOption], PTCPEnd
PTCP-FollowUpPDU	PTCPSubdomain, PTCPTime, [PTCPOption], PTCPEnd
PTCP-DelayReqPDU	PTCPDelayParameter, [PTCPOption], PTCPEnd
PTCP-DelayResPDU	PTCPDelayParameter, PTCPPortParameter, PTCPPortTime, [PTCPOption], PTCPEnd
PTCP-DelayFuResPDU	PTCPDelayParameter, [PTCPOption], PTCPEnd
[a] Legacy support	

Table 146 defines structures for substitutions of elements of the APDU structures.

Table 146 – PTCP substitutions

Substitution name	Structure
PTCP-Header-Sync	PTCP_reserved_1, PTCP_reserved_2, PTCP_Delay10ns, PTCP_SequenceID, PTCP_Delay1ns_Byte [d], Padding [a], PTCP_Delay1ns
PTCP-Header-FUP	[HWPadding*] [b], PTCP_SequenceID, [Padding*] [c], PTCP_Delay1ns_FUP
PTCP-Header-Announce	[HWPadding*] [b], PTCP_SequenceID, [Padding*] [e]
PTCP-Header-Delay	[HWPadding*] [b], PTCP_SequenceID, [Padding*] [c], PTCP_Delay1ns
PTCPSubdomain	PTCP_TLVHeader, PTCP_MasterSourceAddress, PTCP_SubdomainUUID
PTCPTime	PTCP_TLVHeader, PTCP_EpochNumber, PTCP_Seconds, PTCP_NanoSeconds
PTCPTimeExtension	PTCP_TLVHeader, PTCP_Flags, PTCP_CurrentUTCOffset, [Padding*] [a]
PTCPMaster	PTCP_TLVHeader, PTCP_MasterPriority1, PTCP_MasterPriority2, PTCP_ClockClass, PTCP_ClockAccuracy, PTCP_ClockVariance, [Padding*] [a]
PTCPDelayParameter	PTCP_TLVHeader, PTCP_PortMACAddress
PTCPPortParameter	PTCP_TLVHeader, [Padding*] [a], PTCP_T2PortRxDelay, PTCP_T3PortTxDelay
PTCPPortTime	PTCP_TLVHeader, [Padding*] [a], PTCP_T2TimeStamp
PTCPOption	PTCP_TLVHeader, PTCP_OUI, PTCP_SubType, Data*, [Padding*] [a]
PTCPEnd	PTCP_TLVHeader(0)
[a] 32 bit alignment shall be ensured.	
[b] 12 octets padding.	
[c] 2 octets padding.	
[d] Legacy to cover hardware needs.	
[e] 6 octets padding.	

4.4.1.3 Coding section related to common basic fields

4.4.1.3.1 Coding of the field PTCP_reserved_1

This field shall be coded as data type Unsigned32.

4.4.1.3.2 Coding of the field PTCP_reserved_2

This field shall be coded as data type Unsigned32.

4.4.1.3.3 Coding of the field HWPadding

This field shall be coded as data type Unsigned8.

4.4.1.3.4 Coding of the field PTCP_TLVHeader

The coding of this field shall be according to 3.4.2.3.4. and the individual bits shall have the following meaning:

Bit 0 – 8: PTCP_TLVHeader.Length
The value contains the sum of subsequent octets of the respective block.

Bit 9 – 15: PTCP_TLVHeader.Type
This field shall be coded with the values according to Table 147.

Table 147 – PTCP_TLVHeader.Type

Value (hexadecimal)	Meaning	Use
0x00	PTCPEnd	Mandatory
0x01	PTCPSubdomain	Mandatory
0x02	PTCPTime	Mandatory
0x03	PTCPTimeExtension	Mandatory
0x04	PTCPMaster	Mandatory
0x05	PTCPPortParameter	Mandatory
0x06	PTCPDelayParameter	Mandatory
0x07	PTCPPortTime	Mandatory
0x08 – 0x7E	Reserved	—
0x7F	PTCPOption organization-specific	Optional

4.4.1.3.5 Coding section related to PTCP-Header

4.4.1.3.5.1 General

These fields contain the propagation delay time in nanoseconds. They are divided into a field for the 10 ns part and a field for the 1 ns part. All delay fields are relevant only for Sync-, FollowUp-, DelayRes and DelayFuRes-Messages.

4.4.1.3.5.2 Coding of the field PTCP_Delay10ns

This field shall be coded as data type Unsigned32 and shall contain the 10 ns part of the propagation delay time. This field shall be coded with the values according to Table 148.

Table 148 – PTCP_Delay10ns

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFE	Valid 10 nanosecond part of the propagation delay time.
0xFFFFFFFF	Invalid This value indicates an overflow and marks the delay fields as invalid.

4.4.1.3.5.3 Coding of the field PTCP_Delay1ns_Byte

The field contains the 1 ns part of the propagation delay time.

NOTE Legacy, to cover hardware needs.

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0 – 3: PTCP_Delay1ns_Byte.Value
The value contains the 1 ns part of the propagation delay time. This field shall be coded with the values according to Table 149.

Table 149 – PTCP_Delay1ns_Byte.Value

Value (hexadecimal)	Meaning
0x00 – 0x09	Number of ns
0x0A – 0x0F	Reserved

Bit 4 – 7: PTCP_Delay1ns_Byte.Reserved
This field shall be set according to 3.4.2.2.

Instead of this field only the field PTCP_Delay1ns may be used.

4.4.1.3.5.4 Coding of the field PTCP_Delay1ns

This field shall be coded as data type Unsigned32 and shall contain the 1 ns part of the propagation delay time. This field shall be coded with the values according to Table 150.

Table 150 – PTCP_Delay1ns

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFE	Valid 1 nanosecond part of the propagation delay time.
0xFFFFFFFF	Invalid This value indicates an overflow and marks the delay fields as invalid.

4.4.1.3.5.5 Coding of the field PTCP_Delay1ns_FUP

This field shall be coded as data type Integer32 and shall contain the correction of the propagation delay time in 1 ns. This field shall be coded with the values according to Table 151.

Table 151 – PTCP_Delay1ns_FUP

Value (decimal)	Meaning
-2 147 483 648 – 2 147 483 647	Nanosecond part of the propagation delay time.

4.4.1.3.5.6 Coding of the field PTCP_SequenceID

This field shall be coded as data type Unsigned16 with the values according to Table 152.

Table 152 – PTCP_SequenceID

Meaning	Usage
Sync frame	The PTCP master increments this field for every sync frame.
Follow up frame	The PTCP master mirrors the value from the sync frame. If the PTCP forwarder needs to inject a follow up frame, the PTCP forwarder mirrors the value from the sync frame.
Delay request frame	The PTCP delay measurement requester increments this field for every measurement.
Delay response frame	The responder mirrors the value from the Delay request frame.
Delay response follow up frame	The responder uses the value from the Delay response frame.

The receiver of the PTCP frames shall use this field to detect repetitions, loss of frames and timeliness of data. For this reason, the PTCP_SequenceID value range shall be divided in the ratio 15/16 for valid and 1/16 for invalid as shown in Figure 36.

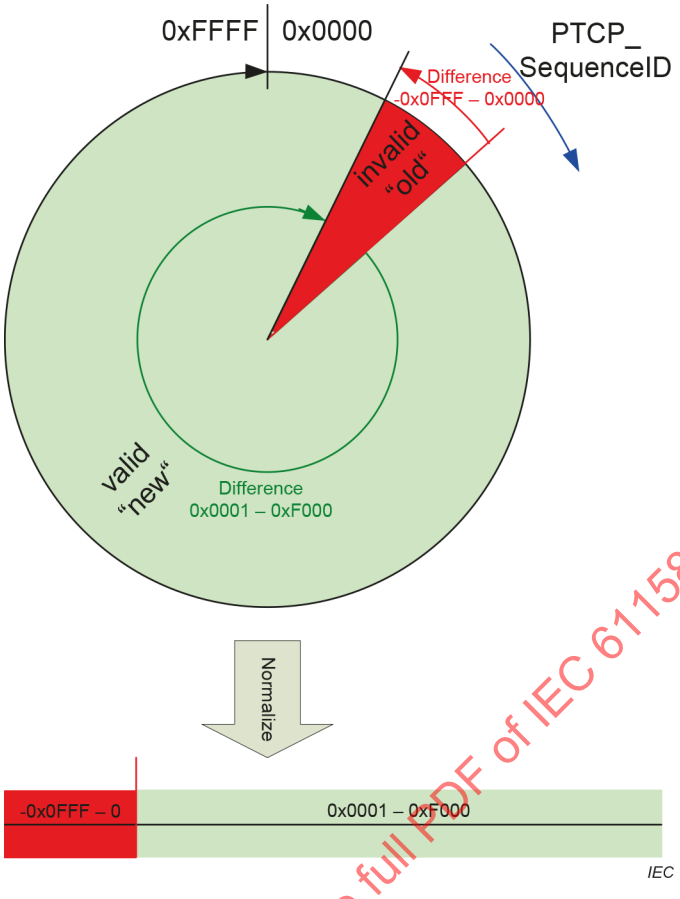


Figure 36 – PTCP_SequenceID value range

4.4.1.3.6 Coding of the field PTCP_OUI

This field shall be coded as OctetString[3] with the Organizationally Unique Identifier (OUI) as defined in IEEE Std 802.

NOTE The value of this central administrative number is given by the IEEE Registration Authority Committee. Available at <standards.ieee.org/regauth>

4.4.1.3.7 Coding of the field PTCP_SubType

This field shall be coded as Unsigned8. The value is organization specific. For the OUI the parameter PTCP_SubType shall be coded as in Table 153.

Table 153 – PTCP_SubType for OUI (=00-0E-CF)

Value (hexadecimal)	Meaning	Usage
0x00 – 0xFF	Reserved	—

4.4.1.3.8 Coding of the field PTCP_Seconds

This field shall be coded as data type Unsigned32 with the values according to Table 154. The resolution of this field is 1 s.

Table 154 – PTCP_Seconds

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Time in s

4.4.1.3.9 Coding of the field PTCP_NanoSeconds

This field shall be coded as data type Unsigned32 with the values according to Table 155. The resolution of this field is 1 ns.

Table 155 – PTCP_NanoSeconds

Value (hexadecimal)	Meaning
0x00000000 – 0x3B9AC9FF	Time in ns
0x3B9ACA00 – 0xFFFFFFFF	Reserved

4.4.1.4 Coding section of PTCP-PDU specific fields

4.4.1.4.1 Coding of the field PTCP_MasterSourceAddress

This field shall be coded as data type OctetString[6]. The value of the field Destination Address shall be according to the 48-bit universal LAN MAC addresses of IEEE Std 802.

4.4.1.4.2 Coding of the field PTCP_SubdomainUUID

This field shall be coded as data type UUID.

NOTE The value NIL indicates no synchronization within the Read Real Sync Data service.

4.4.1.4.3 Coding of the field PTCP_Flags

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: PTCP_Flags.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 8 – 9: PTCP_Flags.LeapSecond
This field shall be set according to Table 156.

Table 156 – PTCP_Flags.LeapSecond

Value (hexadecimal)	Meaning
0x00	No leap second
0x01	Last minute has 61 s
0x02	Last minute has 59 s
0x03	Reserved

Bit 10 – 15: PTCP_Flags.Reserved_2
This field shall be set to zero.

4.4.1.4.4 Coding of the field PTCP_EpochNumber

This field shall be coded as data type Unsigned16 and contain the 0x100000000 s part of the time. The value of the PTCP_EpochNumber field shall be the value of epoch number of the time properties data set of the WorkingClock issuing this message according to Table 157.

When used as WorkingClock, the timescale is arbitrary.

Table 157 – Timescale correspondence between PTCP_EpochNumber, PTCP_Second, PTCP_Nanosecond, CycleCounter and SendClockFactor

CycleCounter value (hexadecimal)	PTCP_EpochNumber value (decimal)	PTCP_Second value (decimal)	PTCP_Nanosecond value (decimal)	SendClockFactor value (decimal)
0x0000000000000000	0	0	0	32
0x0000000000010000	0	65	536 000 000	32
0x0000000100000000	0	4 294 967	296 000 000	32
0x0001000000000000	65	2 302 102 470	656 000 000	32

$$PTCP\_Epoch = PTCP\_EpochNumber \times 0x100000000 \quad (11)$$

where

PTCP_Epoch is the auxiliary variable for the calculation of seconds

PTCP_EpochNumber is the epoch number

$$PTCP\_Time = PTCP\_Epoch + PTCP\_Second + PTCP\_Nanosecond \quad (12)$$

where

PTCP_Time is the PTCP time

PTCP_Epoch is the auxiliary variable for the calculation of seconds

PTCP_Second is the PTCP second value

PTCP_Nanosecond is the PTCP nanosecond value

4.4.1.4.5 Coding of the field PTCP_CurrentUTCOffset

This field shall be coded as data type Integer16 with the values according to Table 158.

Table 158 – PTCP_CurrentUTCOffset

Value (decimal)	Usage
0	Default
Other	Reserved

4.4.1.4.6 Coding of the field PTCP_MasterPriority1

PTCP_MasterPriority1 is used in the execution of the best master clock algorithm. Lower values take precedence. The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0 – 2: PTCP_MasterPriority1.Priority
This field shall be set according to Table 159 and Table 160.

Table 159 – PTCP_MasterPriority1.Priority for SyncID == 0 and SyncProperties.Role == 2

Value (hexadecimal)	Usage	Meaning
0x00	Reserved	—
0x01	Primary synchronization master	Clock synchronization (primary master)
0x02	Secondary synchronization master	Clock synchronization (secondary master)
0x03 – 0x07	Reserved	

Table 160 – PTCP_MasterPriority1.Priority for SyncID == 0 and SyncProperties.Role == 1

Value (hexadecimal)	Usage	Meaning
0x00	Sync slave	Clock synchronization (slave)
0x01 – 0x07	Reserved	—

Bit 3 – 5: PTCP_MasterPriority1.Level
This field shall be set according to Table 161.

Table 161 – PTCP_MasterPriority1.Level

Value (hexadecimal)	Usage	Meaning
0x00	Level 0 (highest)	See Annex I
0x01	Level 1	
0x02 – 0x06	...	
0x07	Level 7 (lowest)	

Bit 6: PTCP_MasterPriority1.Reserved
This field shall be set to zero.

Bit 7: PTCP_MasterPriority1.Active
This field shall be set to zero for the PDSyncData.

This field shall be set to zero for the PTCP-ANNOUNCE-PDU if the sync master does not convey PTCP-SYNC-PDUs and shall be set to one if the sync master conveys PTCP-SYNC-PDUs.

Thus, this field shall be set to 1 for the PTCP-SYNC-PDU after the best master selection is done.

4.4.1.4.7 Coding of the field PTCP_MasterPriority2

This field shall be coded as data type Unsigned8. PTCP_MasterPriority2 is used in the execution of the best master clock algorithm. Lower values take precedence. The value of PTCP_MasterPriority2 shall be configurable according to Table 162.

Table 162 – PTCP_MasterPriority2

Value (hexadecimal)	Usage	Meaning
0x00 – 0xFE	Reserved	—
0xFF	Default	Default

4.4.1.4.8 Coding of the field PTCP_ClockClass

This field shall be coded as data type Unsigned8 with the values according to Table 163. For working clock synchronization, the arbitrary timescale (ARB) shall be used.

Table 163 – PTCP_ClockClass for SyncID == 0 (working clock synchronization)

Value (hexadecimal)	Usage	Meaning
0x00 – 0x0C	Reserved	—
0x0D	Shall designate a PTCP clock as a primary reference that is synchronized to an application specific time source. The timescale distributed shall be ARBITRARY. A clock class 0x0D shall not be a slave to another clock in the PTCP domain.	Default
0x0E	Shall designate a PTCP clock that has previously been designated as clock class 0x0D but that has lost the ability to synchronize to an application specific time source and is in holdover mode and within holdover specifications. The timescale distributed shall be ARBITRARY. A clock class 0x0E shall not be a slave to another clock in the PTCP domain.	—
0x0F – 0xBA	Reserved	—
0xBB	Degradation alternative B for a PTCP clock of clock class 0x07 that is not within holdover specification. A clock of clock class 0xBB may be a slave to another clock in the PTCP domain.	—
0xBC – 0xC0	Reserved	—
0xC1	Degradation alternative B for a PTCP clock of clock class 0x0E that is not within holdover specification. A clock of clock class 0xC1 may be a slave to another clock in the PTCP domain.	—
0xC2 – 0xFE	Reserved	—
0xFF	Shall be the clock class of a slave-only clock.	—

4.4.1.4.9 Coding of the field PTCP_ClockAccuracy

This field shall be coded as data type Unsigned8 with the values according to Table 164. The PTCP_ClockAccuracy indicates the expected accuracy of a clock when it is master. The value of PTCP_ClockAccuracy shall represent its status.

Table 164 – PTCP_ClockAccuracy

Value (hexadecimal)	Usage	Meaning
0x00 – 0x1F	Reserved	—
0x20	The time is accurate up to 25 ns	—
0x21	The time is accurate up to 100 ns	—
0x22	The time is accurate up to 250 ns	—
0x23	The time is accurate up to 1 µs	—
0x24	The time is accurate up to 2,5 µs	—

Value (hexadecimal)	Usage	Meaning
0x25	The time is accurate up to 10 µs	—
0x26	The time is accurate up to 25 µs	—
0x27	The time is accurate up to 100 µs	—
0x28	The time is accurate up to 250 µs	—
0x29	The time is accurate up to 1 ms	—
0x30 – 0xFD	Reserved	—
0xFE	Time accuracy unknown	Default for working clock synchronization
0xFF	Reserved	—

4.4.1.4.10 Coding of the field PTCP_ClockVariance

This field shall be coded as data type Unsigned16 with the values according to Table 165. The value characterizes the quality of a clock. Each master clock shall maintain an estimate of its inherent precision. This is the precision of the timestamps included in the sync-frames issued by the PTCP synchronization master when it is not synchronized to another PTCP synchronization master using the PTCP protocol.

Table 165 – PTCP_ClockVariance

Value (hexadecimal)	Usage	Meaning
0x0000	Reserved	Default
0x0001 – 0xFFFF	Reserved	—

4.4.1.4.11 Coding of the field PTCP_T2PortRxDelay

This field shall be coded as data type Unsigned32 with the values according to Table 166.

Table 166 – PTCP_T2PortRxDelay

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Port receive delay (MDI to SHIM) in nanoseconds.

4.4.1.4.12 Coding of the field PTCP_T3PortTxDelay

This field shall be coded as data type Unsigned32 with the values according to Table 167.

Table 167 – PTCP_T3PortTxDelay

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Port transmit delay (SHIM to MDI) in nanoseconds.

4.4.1.4.13 Coding of the field PTCP_PortMACAddress

This field shall be coded as data type OctetString[6]. The value of the field PTCP_PortMACAddress shall be according to the 48-bit universal LAN MAC addresses of IEEE Std 802.

This field is used to identify the relation between PTCP-DelayReqPDU, PTCP-DelayResPDU, and PTCP-DelayFuResPDU. The PTCP-DelayResPDU and PTCP-DelayFuResPDU mirror the value from the PTCP-DelayReqPDU.

4.4.1.4.14 Coding of the field PTCP_T2TimeStamp

This field shall be coded as data type Unsigned32 with the values according to Table 168.

Table 168 – PTCP_T2TimeStamp

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	TimeStamp in nanoseconds

4.4.2 AP-Context state machine

There is no AP-Context State Machine defined for this Protocol.

4.4.3 FAL Service Protocol Machines

There is no FAL Service Protocol Machine (FSPM) defined for this Protocol.

4.4.4 Application Relationship Protocol Machines

4.4.4.1 Generic Synchronization with PTCP

4.4.4.1.1 Timestamp

The PTCP message timestamp point for a transmitted or received frame shall correspond to the leading edge of the first bit of the octet immediately following the Start Frame Delimiter octet of an IEEE Std 802.3 frame as shown in Figure 37.

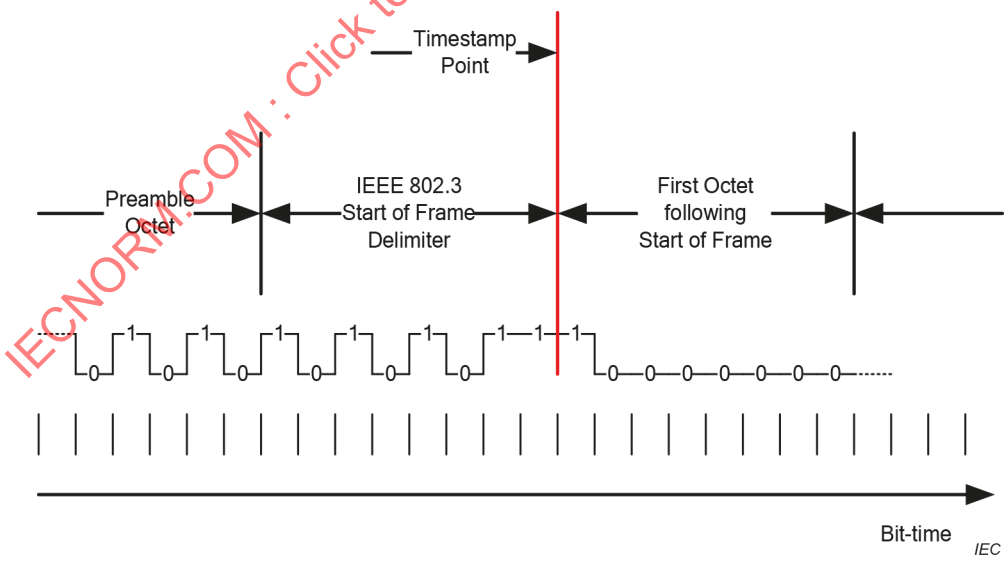


Figure 37 – Message timestamp point

4.4.4.1.2 Timer model

PTCP uses the timer model shown in Figure 38.

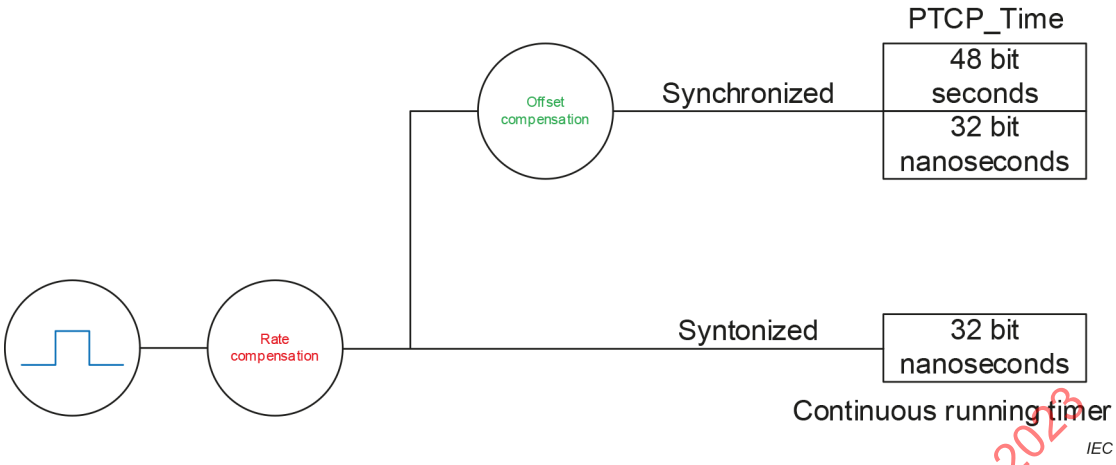


Figure 38 – Timer model

The timer model consists of two timers. The first, at least 32 bit wide syntonized continuous running timer with a resolution of one nanosecond, is used for the line delay and bridge delay measurement, the second, 80 bit wide synchronized timer with a resolution of one nanosecond is used for the PTCP_Time of master or slave.

4.4.4.1.3 Generic model

For each received and transmitted sync, delay request or delay response message a time stamping unit shall generate a time stamp. Figure 39 shows the four timestamps used for synchronization.

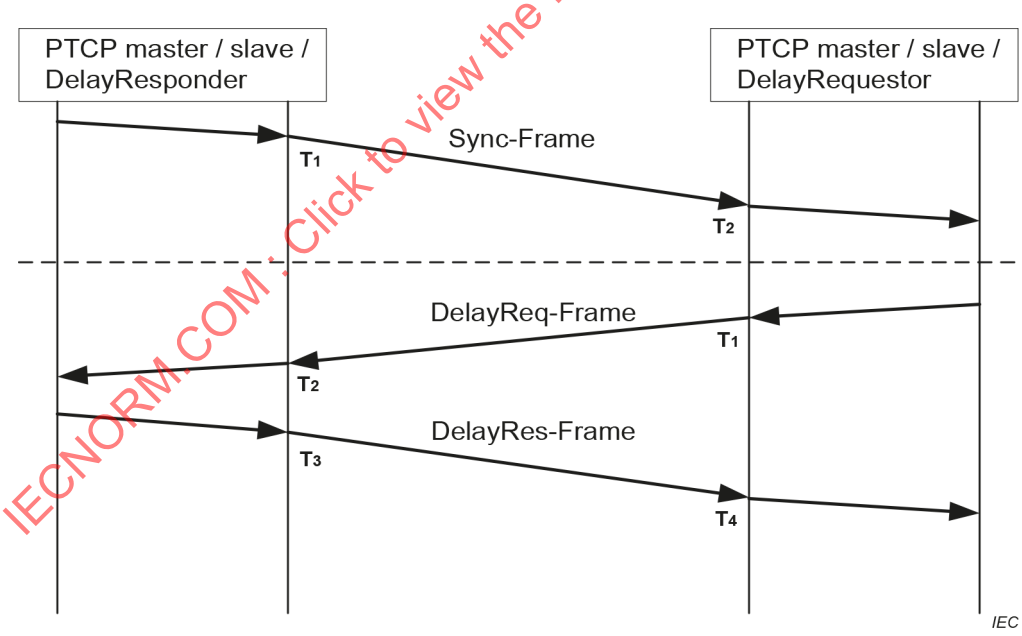


Figure 39 – Four message timestamps

The generic model uses the following definitions:

$$SendTimeStamp = T_1$$

$$ReceiptTimeStamp = T_2$$

$$SendTimeStamp = T_3$$

$$ReceiptTimeStamp = T_4$$

where

T_1 is measured by the local clock of the port which sends a Sync- or a DelayReq-Frame

T_2 is measured by the local clock of the port which receives a Sync- or a DelayReq-Frame

T_3 is measured by the local clock of the port which sends a DelayRes-Frame

T_4 is measured by the local clock of the port which receives a DelayRes-Frame

4.4.4.1.4 Line delay measurement and peer rate compensation

The line delay measurement and the peer rate compensation calculation between DelayRequestor and DelayResponder are described in Figure 40. DelayReq-, DelayRes and DelayFuRes-Messages shall be used for delay measurement and shall not be propagated. The DelayFuRes-Frame is used to transmit the value of ResDelay to the DelayRequestor.

DelayResponders that are not able to measure the line delay but want to receive the Sync-Frame shall use a DelayRes.req with a ResDelay value of zero.

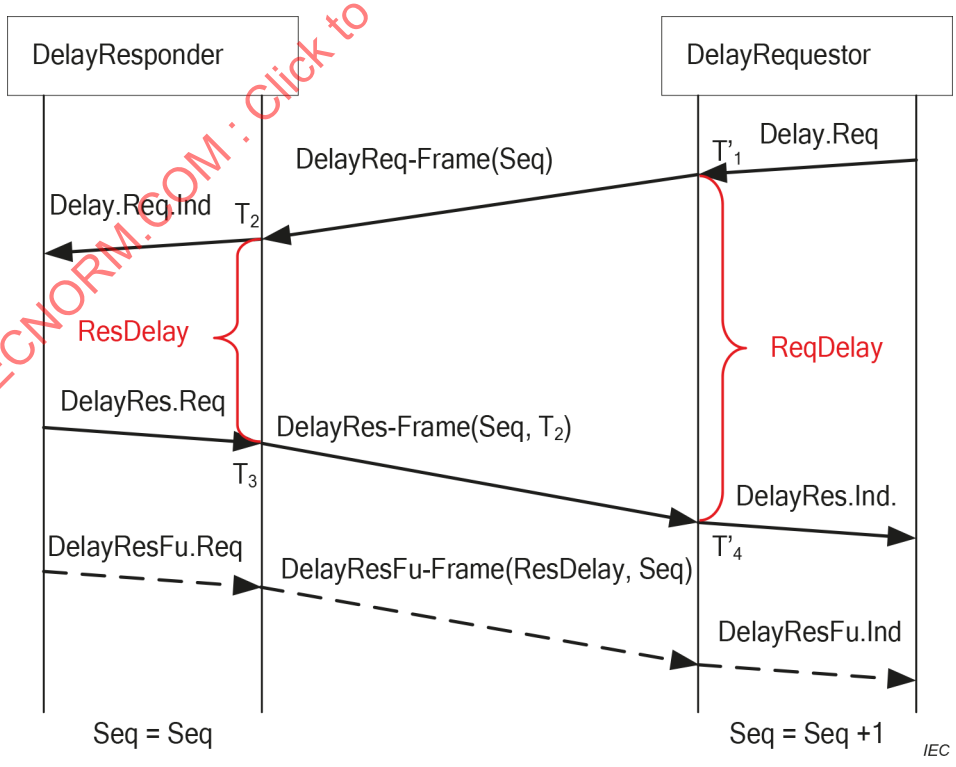


Figure 40 – Line delay protocol with follow up

If a time stamping unit is capable to calculate the response delay time instantly and insert the response time in the delay response frame, no delay follow up message is necessary.

The line delay measurement without DelayFuRes-message between DelayRequestor and DelayResponder is described in Figure 41.

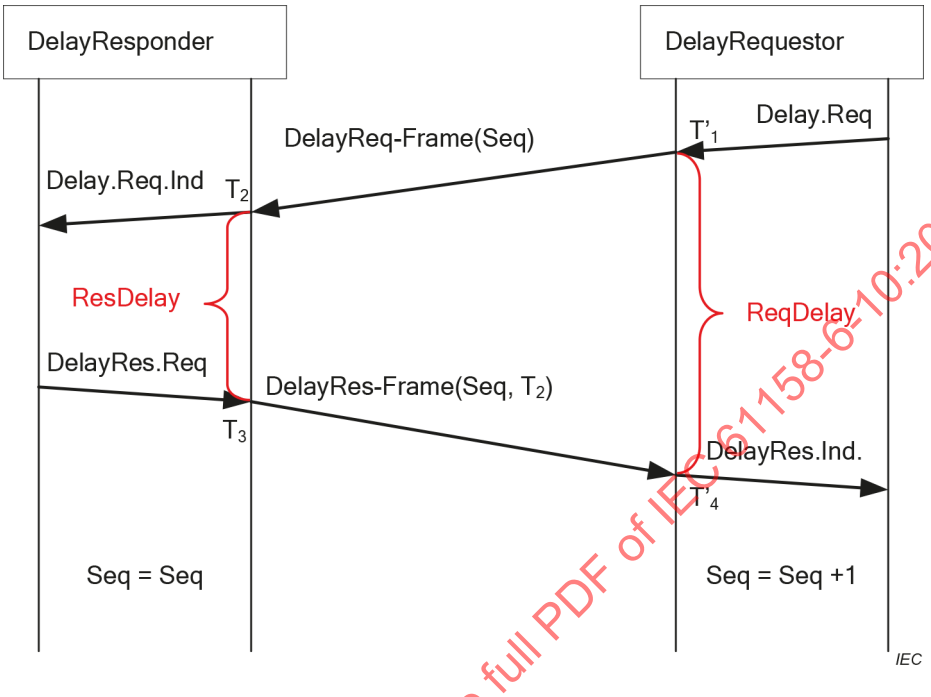


Figure 41 – Line delay protocol without follow up

The calculation of line delay and peer rate compensation factor is shown in the following formulae. If the DelayRes-PDU contains the field T_2, the Formula (15) shall be used, if the ResDelay is zero, RCF_{peer} shall be set to 1.

$$ResDelay = T_3 - T_2 \quad (13)$$

where

ResDelay	is the delay at responder side
T_3	is the time stamp of sending DelayRes-Frame as DelayResponder
T_2	is the time stamp of receiving DelayReq-Frame as DelayResponder

$$ReqDelay = T'_4 - T'_1 \quad (14)$$

where

ReqDelay	delay requestor
T'_4	is the time stamp of receiving DelayRes-Frame as DelayRequestor
T'_1	is the time stamp of sending DelayReq-Frame as DelayRequestor

$$RCF_{peer} = \frac{T_2(Seq) - T_2(Seq-1)}{T'_1(Seq) - T'_1(Seq-1)} \quad (15)$$

where

RCF_{peer}	is the peer rate compensation factor
T_2	is the time stamp of receiving DelayReq-Frame as DelayResponder and transmitted using the DelayRes-Frame to the DelayRequestor
T'_1	is the time stamp of sending DelayReq-Frame as DelayRequestor
Seq	is the sequence variable

$$\text{ResDelaypeer} = \frac{\text{ResDelay}}{\text{RCFpeer}} \tag{16}$$

where

$ResDelay_{peer}$	is the delay responder at this peer
ResDelay	is the delay responder
RCF_{peer}	is the peer rate compensation factor

The delay measurement shall be repeated in cycles.

4.4.4.1.5 Line delay calculation

The line delay between two devices supporting PTCP is determined as described in Figure 42.

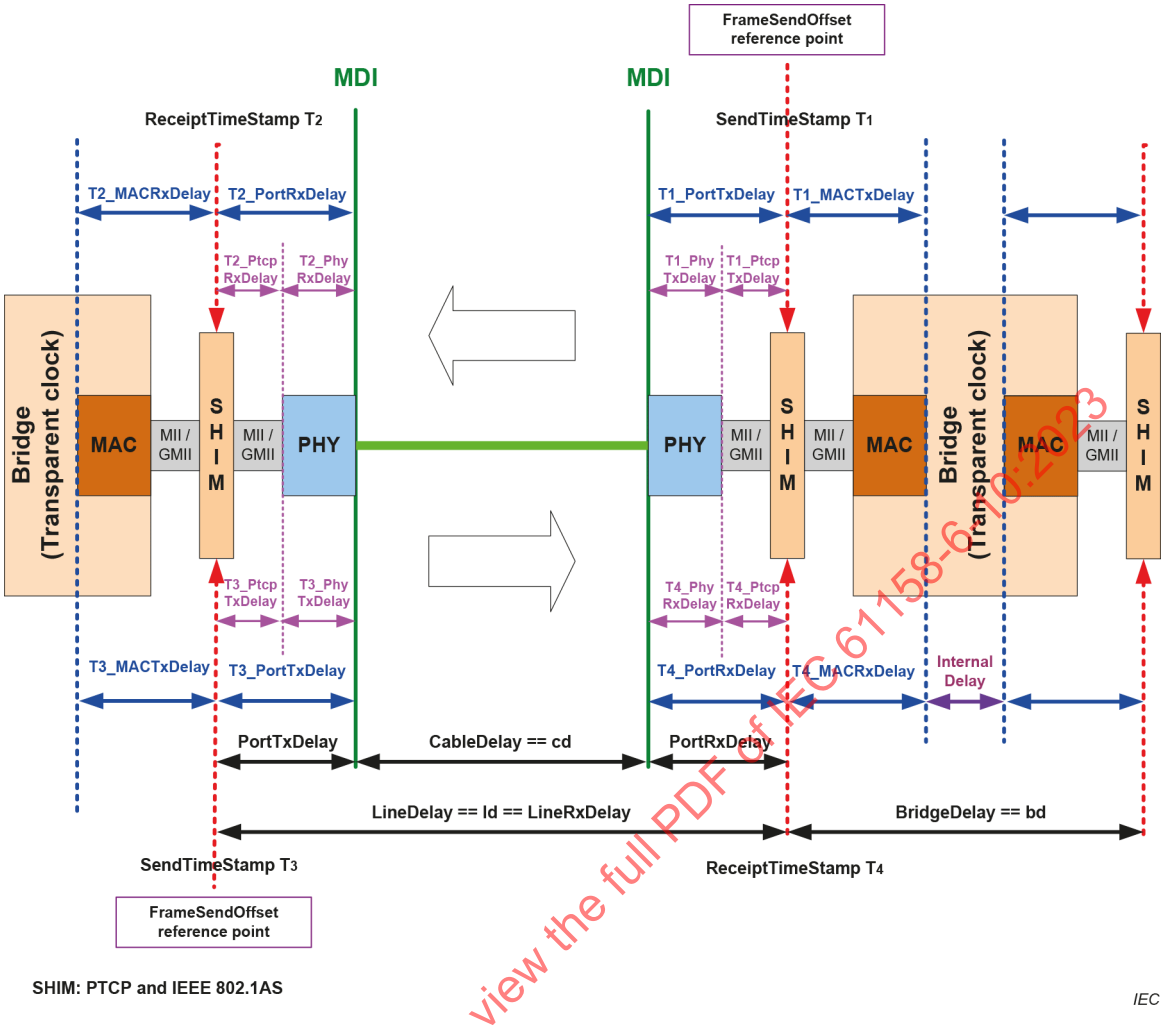


Figure 42 – Line delay measurement

NOTE 1 The Reduced Media Independent Interface (RMII) synchronizes its internal 50 MHz clock with the 25 MHz clock of the PHY, which results in a quantization error of 20 ns. This jitter cannot be compensated. The Media Independent Interface (MII) uses the PHY clock without this error.

NOTE 2 The Class D cable type has a maximal delay skew of 45 ns for 100 m cable length. This results in a 45 ns offset error which cannot be compensated. A selection of better cable reduces this error.

A port receive/transmit delay (example: T1_PortTxDelay) contains the line delay of the used PHY component and the deviation from the reference time stamp.

$$T1\_PortTxDelay = T1\_PhyTxDelay + T1\_PtcpTxDelay \quad (17)$$

where

T1_PortTxDelay is the port transmit delay

T1_PhyTxDelay is the delay of the used PHY component

T1_PtcpTxDelay is the deviation from the reference time stamp

$$T3\_PortTxDelay = T3\_PhyTxDelay + T3\_PtcpTxDelay \quad (18)$$

where

T3_PortTxDelay	is the port transmit delay
T3_PhyTxDelay	is the delay of the used PHY component
T3_PtcpTxDelay	is the deviation from the reference time stamp

$$T2\_PortRxDelay = T2\_PhyRxDelay + T2\_PtcpRxDelay \quad (19)$$

where

T2_PortRxDelay	is the port receive delay
T2_PhyRxDelay	is the delay of the used PHY component
T2_PtcpRxDelay	is the deviation from the reference time stamp

$$T4\_PortRxDelay = T4\_PhyRxDelay + T4\_PtcpRxDelay \quad (20)$$

where

T4_PortRxDelay	is the port receive delay
T4_PhyRxDelay	is the delay of the used PHY component
T4_PtcpRxDelay	is the deviation from the reference time stamp

$$CableDelay = (ReqDelay - ResDelay_{peer} - T1\_PortTxDelay - T2\_PortRxDelay - T3\_PortTxDelay - T4\_PortRxDelay) / 2 \quad (21)$$

where

CableDelay	is the delay of the cable
ReqDelay	is the delay requestor
ResDelaypeer	is the delay responder at this peer
T1_PortTxDelay	is the port transmit delay
T2_PortRxDelay	is the port receive delay
T3_PortTxDelay	is the port transmit delay
T4_PortRxDelay	is the port receive delay

$$LineDelay = CableDelay + T3\_PortTxDelay + T4\_PortRxDelay \quad (22)$$

where

LineDelay	is the delay between both PTCPSHIMs for received frames
CableDelay	is the delay of the cable
T3_PortTxDelay	is the port transmit delay
T4_PortRxDelay	is the port receive delay

$$LineDelay_{SyncMaster} = LineDelay \times RCF_{SyncMaster} \quad (23)$$

where

LineDelaySyncMaster	is the compensated time needed to forward a frame
LineDelay	is the measured value between two peers
RCFSyncMaster	is the factor between the frequency of the SyncMaster and the frequency of the SyncSlave

NOTE 3 Formula (23) is only informative if the LineDelay is measured with the syntonized timer.

A DelayRes.req with a ResDelay value of zero indicates that the CableDelay and the LineDelay are not measurable, but the responding peer wants to receive the Sync-Frame.

LineSyncDelay is the line delay of the Sync-Frame. Due to the asymmetry of the involved communication path and due to the resolution of the time stamps it is basically impossible to determine the line delay exactly.

A PTCP clock can detect non PTCP neighbors by checking the line delay between nodes. As a switch delay is in the range of several (>2) µs, the delay between two nodes connected by 100 m Twisted Pair cabling (100 Base TX) is about 500 ns.

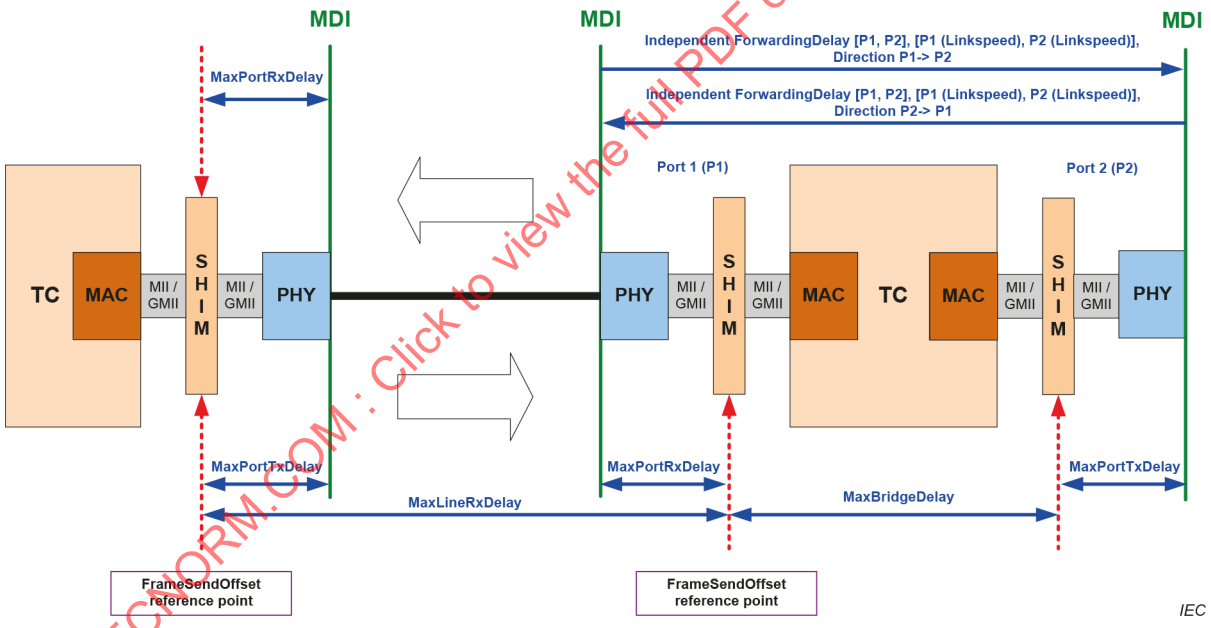


Figure 43 – Model parameter for GSDML usage

Figure 43 shows the model parameter for GSDML usage. The parameter MaxBridgeDelay, MaxPortTxDelay and MaxPortRxDelay offers engineering tools a calculation basis for RT_CLASS_3.

The MaxBridgeDelay to be used depends on the bridging mode, such as normal forwarding, fast forwarding and DFP concatenated forwarding.

4.4.4.1.6 Sync-Frame forwarding

A device which forwards PTCP sync messages measures the bridge delay as shown in Figure 44 and adds this time to the delay field in the sync or follow up message.

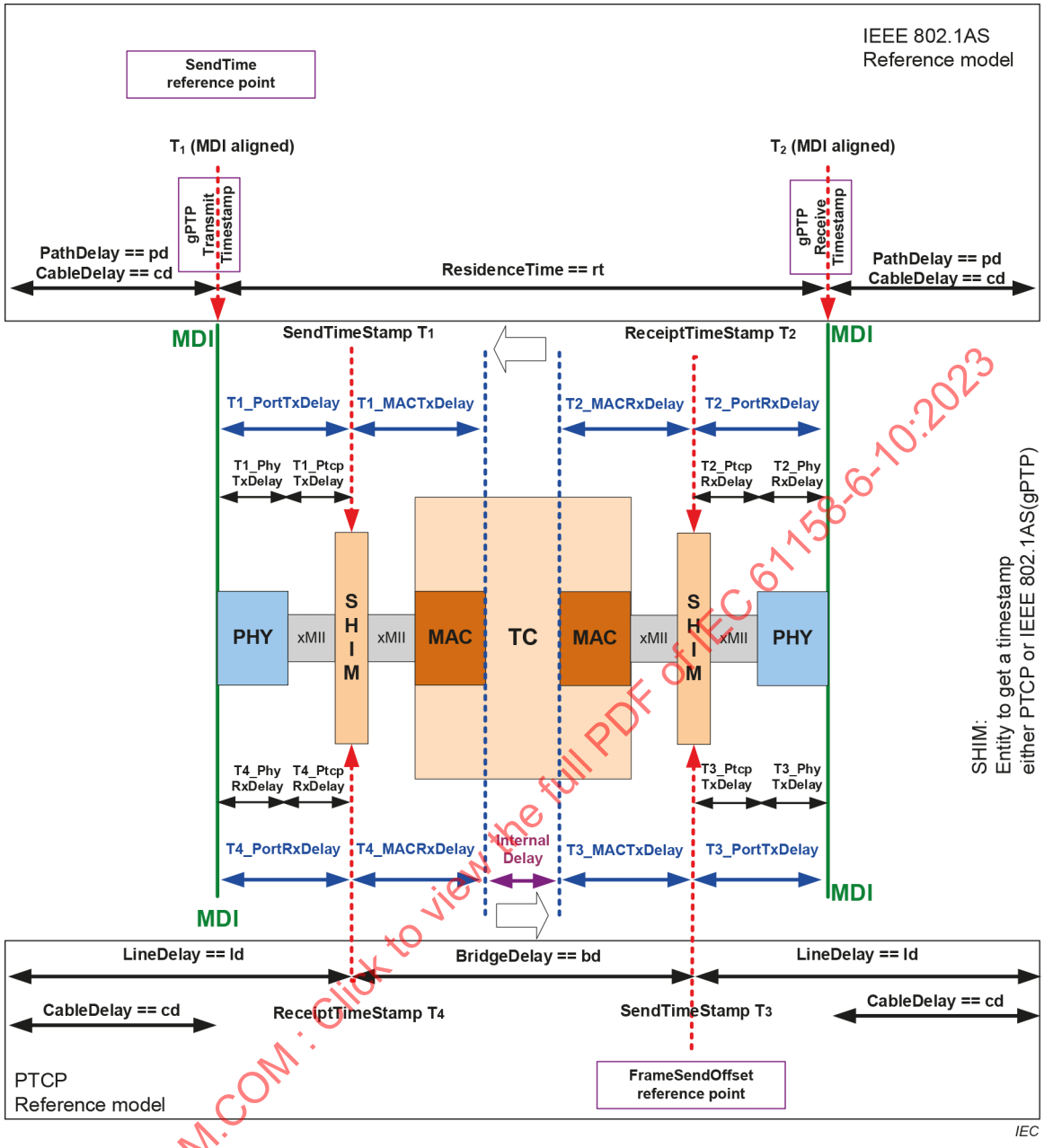


Figure 44 – Bridge delay measurement

The calculation of the residence time is shown in the following formulae.

$$\text{ResidenceTime} = T_1 \text{ (MDI aligned)} - T_2 \text{ (MDI aligned)} \quad (24)$$

where

ResidenceTime	is the time needed to forward a frame
T_1 (MDI aligned)	is the point in time when the first bit of the start delimiter (SD) of a frame is transmitted at MDI
T_2 (MDI aligned)	is the point in time when the first bit of the start delimiter (SD) of a frame is received at MDI

The following calculation model is in principle identical for residence time and bridge delay.

The calculation of the bridge delay is shown in the following formulae.

$$BridgeDelay = T_3 - T_4 \quad (25)$$

where

BridgeDelay	is the time needed to forward a frame
T_3	is the point in time when the first bit of the start delimiter (SD) of a frame is transmitted
T_4	is the point in time when the first bit of the start delimiter (SD) of a frame is received

$$BridgeDelay_{SyncMaster} = BridgeDelay \times RCF_{SyncMaster} \quad (26)$$

where

$BridgeDelay_{SyncMaster}$	is the compensated time needed to forward a frame
BridgeDelay	is the time needed to forward a frame
$RCF_{SyncMaster}$	is the factor between the frequency of the SyncMaster and the frequency of the SyncSlave

NOTE Formula (26) is only informative if the LineDelay is measured with the syntonized timer.

$$RCF_{SyncMaster} = \Delta t_{SyncMaster} / \Delta t_{SyncSlave} \quad (27)$$

where

$RCF_{SyncMaster}$	is the ratio between the clock of the SyncMaster and the SyncSlave calculated by the SyncSlave
$\Delta t_{SyncMaster}$	is a time interval of at least 200 ms from the SyncMaster point of view
$\Delta t_{SyncSlave}$	is a time interval measured at the SyncSlave for the given $\Delta t_{SyncMaster}$

The principle of synchronization across a sequence of devices can be seen in Figure 45 and Figure 46. The node emitting the time frames for synchronization in the network is known as PTCP master. All other nodes that receive and/or pass on these frames are known as PTCP slaves. A PTCP master uses a reference to a local time system or a global time source.

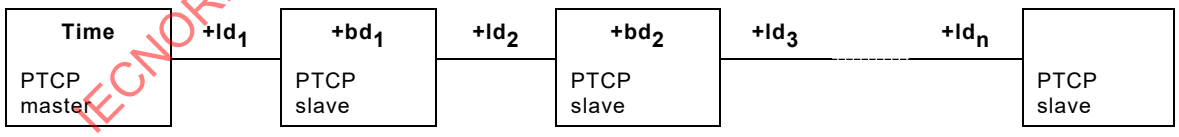


Figure 45 – Delay accumulation for PTCP

The node emitting the time frames for synchronization in the network is known as PTP master. All other nodes that receive and/or pass on these frames are known as PTP slaves. A PTP master uses a reference to a local time system or a global time source.

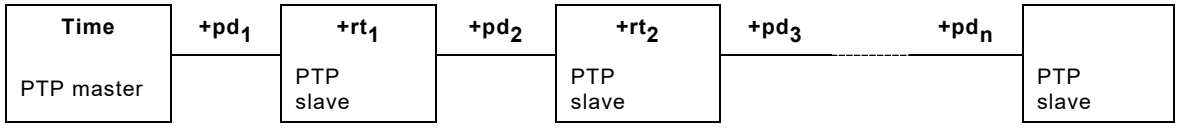


Figure 46 – Delay accumulation for PTP

The Sync-Frames are forwarded as peer-to-peer frames. Every device shall adjust the Sync-Frame to the delay field of the Sync-Frame as it passes through.

The device shall:

– add its bridge delay bd_x (bridge delay measurement is shown in Figure 44) and

– add the line delay ld_x (line delay measurement is shown in Figure 42).

Each PTCP slave knows the propagation delay of the sync frame. When the time of arrival T_4 is known, the drift compared to the PTCP master can also be determined.

4.4.4.1.7 Rate compensation

4.4.4.1.7.1 Bridge delay

The bridge delay should be measured with a timer syntonized / rate compensated to the SyncMaster in order to achieve high precision synchronization.

4.4.4.1.7.2 Line delay measurement

The line delay should be measured with a timer syntonized / rate compensated to the SyncMaster in order to achieve high precision synchronization.

4.4.4.1.8 Time deviation measurement

In order to achieve the measurement of the deviation, see Figure 47, hardware signaling is necessary. For every supported PTCP_SyncID a signal shall be generated. It may only be accessible in test lab environment.

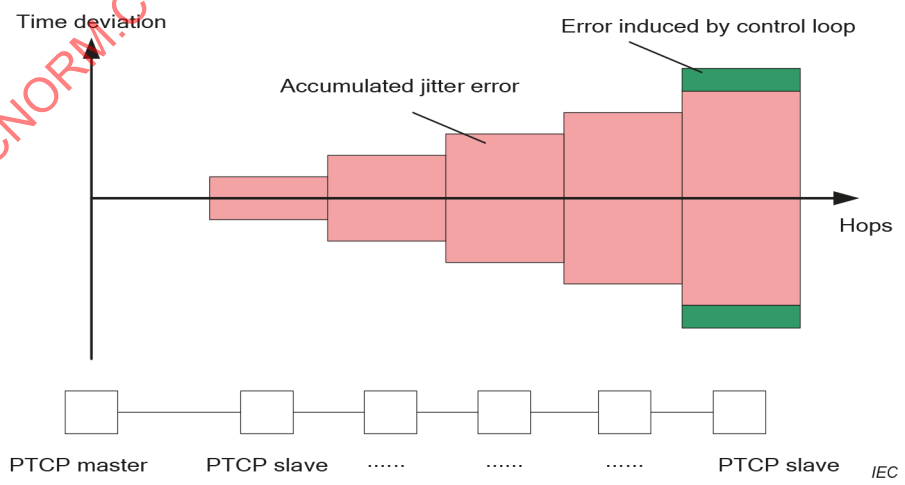


Figure 47 – Worst case accumulated time deviation of synchronization

Figure 48 and Figure 49 show in principle the measurement of the deviation for both, working clock and global time.

Every cycle (SendClock) as shown in Figure 68 shall be signaled for working clock synchronization and every second shall be signaled for global time synchronization.

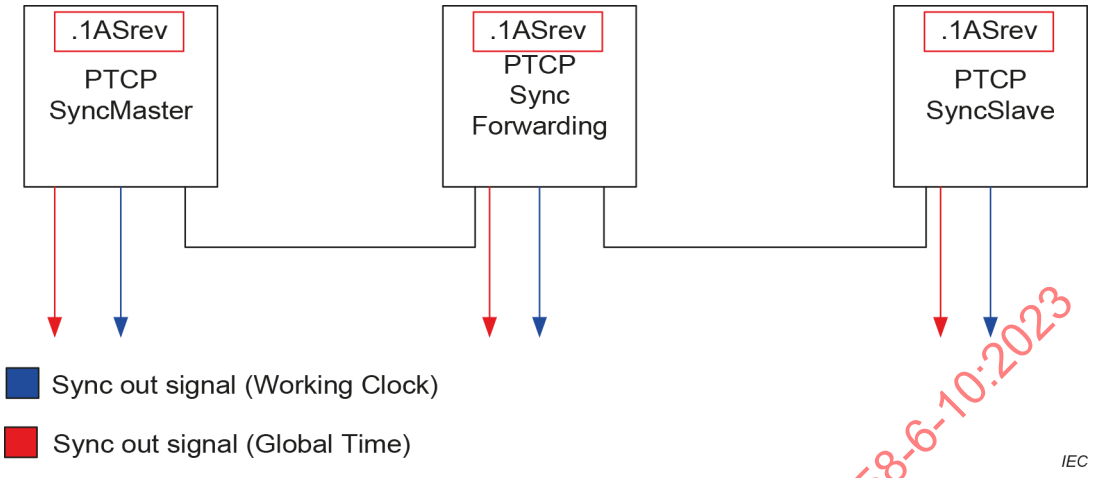


Figure 48 – Signal generation for measurement of deviation

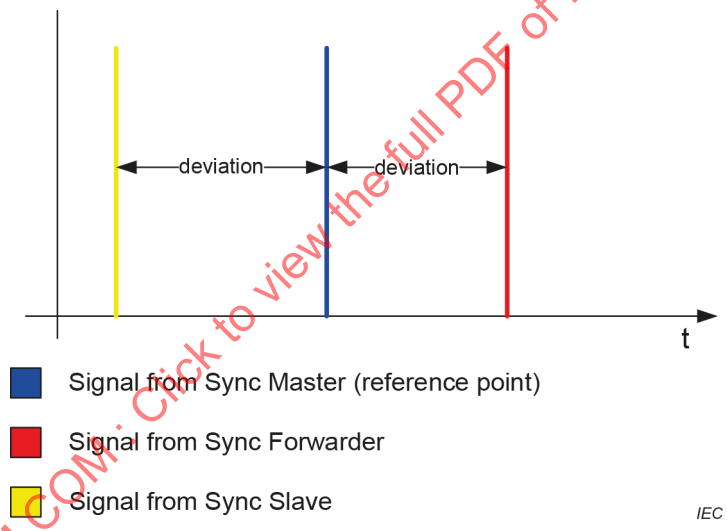


Figure 49 – Measurement of deviation

4.4.4.1.9 Synchronization Protocol

4.4.4.1.9.1 General

PTCP masters send Sync-Frames periodically. The sync frame has an original send time (T_{Org}).

A PTCP master

– sets the PTCPTime field of the sync frame to T_{Org}, and

– sets the PTCP_Delay fields of the sync frame to its internal handling time; for example, the time which passed between T_{Org} and sending of the sync frame.

A PTCP slave adds to the content of the corresponding PTCP_Delay field in the PTCP-Header-Sync the following terms:

– the line delay (T_{LD}) of the port where the sync frame was received, and

– the bridge delay (T_{BD}); for example, the time to forward the sync frame.

Special case: If the PTCP master or some PTCP slave is not able to directly add the delay times into the sync frame PTCP_Delay fields, it shall generate a so-called Follow-Up frame (FU) and add there its bridge delay to the PTCP_Delay field. The line delay is always added to the PTCP_Delay fields of the sync frame.

All times are always related to the measurement timestamp point; see Figure 37 for details.

Each PTCP slave compares its local clock with the sum of PTCPTime, PTCP_Delay of the sync frame and (if applicable) PTCP_Delay of the Follow-Up frame. The offset between T_{Slave} and T_{Master} can thus be determined. The PTCP slave follows the time of the PTCP master.

4.4.4.1.9.2 PTCP Master Synchronization with Sync-Frame

PTCP masters send Sync-Frames periodically. The synchronization scheme as described in Figure 50 requires the transfer of the exact send time of the Sync-Frame as initial value of the delay.

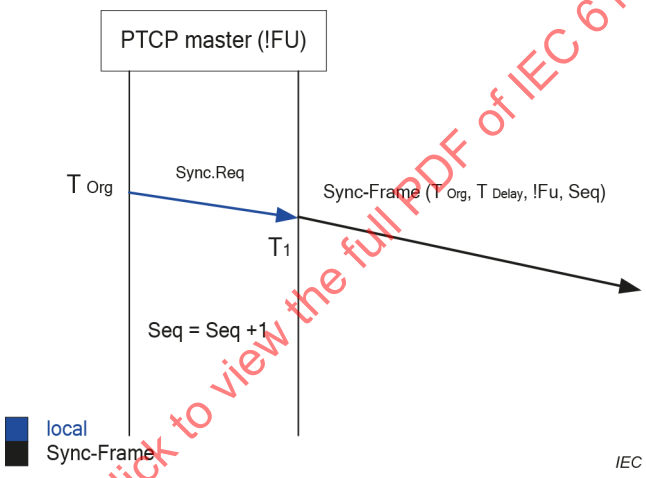


Figure 50 – PTCP master sending Sync-Frame without Follow Up-Frame

$$T_{Delay} = T_1 - T_{Org} \quad (28)$$

where

T_{Delay} is the internal handling time of the sync frame within the PTCP master; it is the value of the PTCP_Delay field within the sync frame sent by the PTCP master

T_1 is the send time stamp of the Sync-Frame by the PTCP master

T_{Org} is the original time of the PTCP master clock

4.4.4.1.9.3 PTCP Master Synchronization with Sync- and FollowUp-Frame

Figure 51 shows a PTCP master which cannot modify a frame before transmission. A PTCP master that is not able to enter the internal handling time directly into the Sync-Frame will use a FollowUp-Frame.

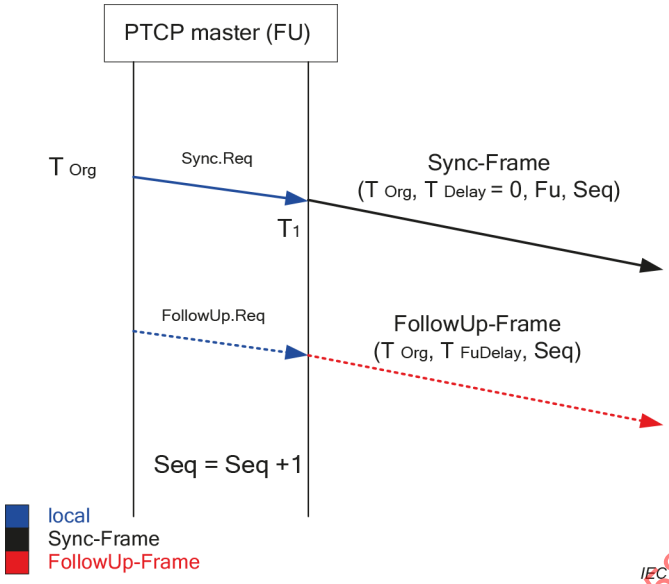


Figure 51 – PTCP master sending Sync-Frame with FollowUp-Frame

The delay field in the Sync-Frame shall be set to zero by the PTCP master. The delay ($T_{FuDelay}$) in the associated FollowUp-Frame contains the initial value of the sync frame's delay.

$$T_{FuDelay} = T_1 - T_{Org} \quad (29)$$

where

$T_{FuDelay}$ is the internal handling time of the sync frame within the PTCP master; it is the value of the PTCP_Delay field within the FollowUp frame sent by the PTCP master.

T_{Org} is the original time of the PTCP master clock

T_1 is the send time stamp of the Sync-Frame by the PTCP master

4.4.4.1.9.4 PTCP Slave Synchronization (!FU)

A PTCP slave as shown in Figure 52 that can modify a frame before transmission (!FU node) adds its internal bridge delay and the line delay to the delay value in the Sync-Frame as it passes through.

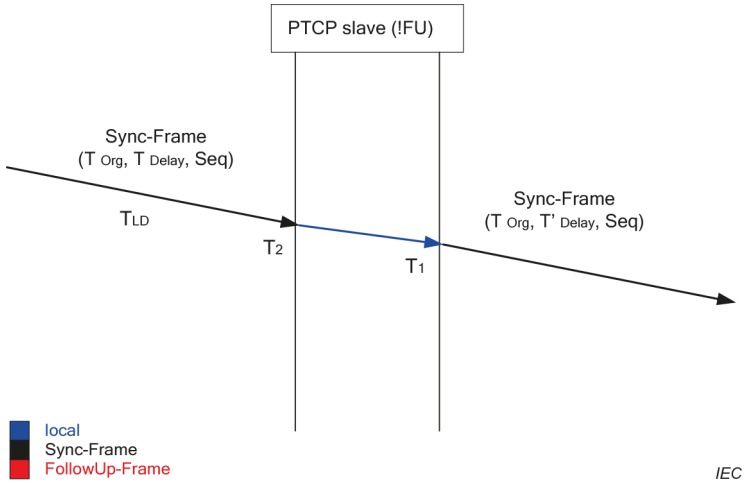


Figure 52 – !FU Sync Slave Forwarding Sync-Frame

$$T'_{Delay} = T_{Delay} + T_{LD} + (T_1 - T_2) \quad (30)$$

where

T'_{Delay}	is the value of the PTCP_Delay field within the sync frame sent by the PTCP slave
T_{Delay}	is the delay time as received by the PTCP slave
T_{LD}	is the line delay time of the receiving port
T_1	is the send time stamp of the Sync-Frame by the PTCP slave
T_2	is the receive time stamp of the Sync-Frame by the PTCP slave

If the sync frame has a FollowUp frame, it is forwarded without any modification by a !FU node.

4.4.4.1.9.5 PTCP Slave Synchronization (FU)

Some implementations of IEEE Std 802.3 do not allow modifying a frame during transmission.

Such a PTCP slave (FU-Node) as shown in Figure 53 adds the line delay to the delay value in the sync frame but adds its internal bridge delay to the delay value in the FollowUp-Frame.

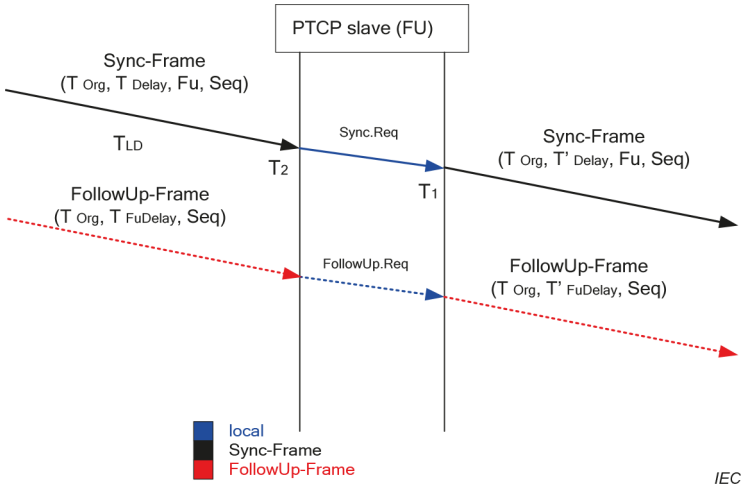


Figure 53 – FU Sync Slave Forwarding Sync- and FollowUp-Frame

$$T'_{Delay} = T_{Delay} + T_{LD} \quad (31)$$

where

T'_{Delay} is the value of the PTCP_Delay field within the sync frame sent by the PTCP slave

T_{Delay} is the delay time within the sync frame as received by the PTCP slave

T_{LD} is the line delay of the receiving port

$$T'_{FuDelay} = T_{FuDelay} + (T_1 - T_2) \quad (32)$$

where

$T'_{FuDelay}$ is the value of the PTCP_Delay field within the FollowUp frame sent by the PTCP slave

$T_{FuDelay}$ is the delay time within the FollowUp frame as received by the PTCP slave

T_1 is the send time stamp of the Sync-Frame by the PTCP slave

T_2 is the receive time stamp of the Sync-Frame by the PTCP slave

If no FollowUp frame exists, it will be generated by an FU PTCP slave. This principle is shown in Figure 54.

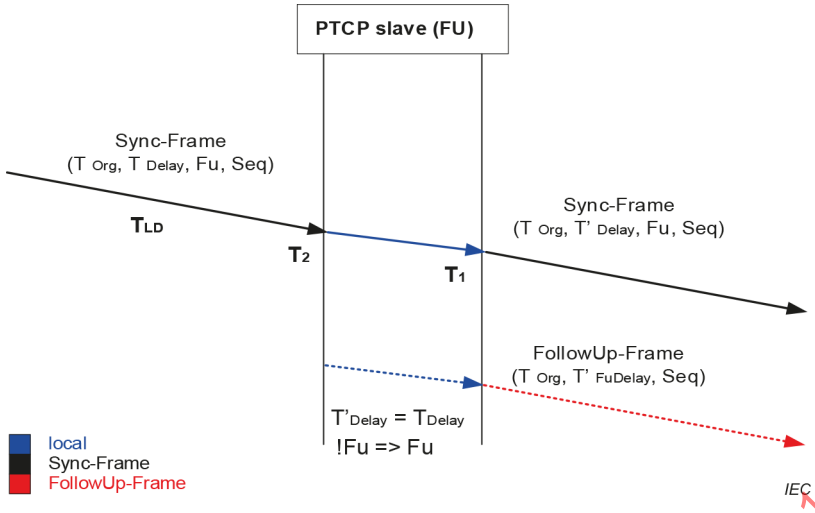


Figure 54 – FU Sync Slave Forwarding Sync- and Generating FollowUp-Frame

$$T'_{Delay} = T_{Delay} + T_{LD} \quad (33)$$

where

T'_{Delay} is the value of the PTCP_Delay field within the sync frame sent by the PTCP slave

T_{Delay} is the delay time within the sync frame as received by the PTCP slave

T_{LD} is the line delay of the receiving port

$$T'_{FuDelay} = (T_1 - T_2) \quad (34)$$

where

$T'_{FuDelay}$ is the value of the PTCP_Delay field within the FollowUp frame sent by the PTCP slave

T_1 is the send time stamp of the Sync-Frame by the PTCP slave

T_2 is the receive time stamp of the Sync-Frame by the PTCP slave

4.4.4.1.10 Error recovery

A link down will delete the line delay. A line delay measurement error will stop the forwarding of all sync messages.

In order to deal with redundancy switchover in case of redundant paths, a Sync-Frame shall be forwarded only if the sequence number difference compared to the previous forwarded message is positive. This prevents duplicates which can corrupt the follow up sequence.

The use of an alternative path is possible as long as there is a valid line delay measurement available.

There are 2 timeouts:

– Delay measurement:

  Allowed time between delay request and delay response

– Monitoring of sync:

  Allowed time between two subsequent sync frames

The timeout of the line delay measurement shall be greater than the timeout of the time master and is calculated independently at every update of the line delay.

For resource reasons the size of the master data base can be limited. Two masters shall be possible at any time. This is possible as the Best Master Algorithm will cancel the activities of a Sync master.

The line delay values can be restricted for some technologies (for example 100 Base TX has delay values below 1 µs). An error shall be reported, but the reaction is beyond the scope of this specification.

4.4.4.2 Line delay measurement

4.4.4.2.1 Line Delay Request Protocol Machine

Figure 55 shows that the line delay is monitored by the LineT und the DelayT timers. Each time the DelayT expires, a line delay measurement starts. If the LineT expires before the line delay measurement is done, an error counter will be increased.

When the current measurement cycle (5 times) is finished, a break for DelayT (Long) is made.

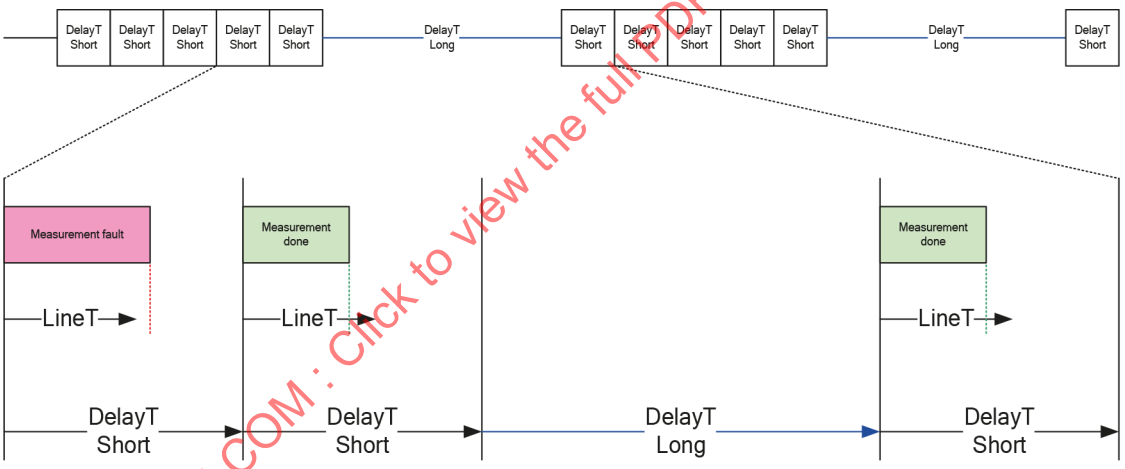


Figure 55 – Principle of the monitoring of the line delay measurement

LineT may be substituted by the checking of the ResDelay value in combination with the DelayT(Short).

4.4.4.2.1.1 Primitive definitions

####### 4.4.4.2.1.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Line Delay Request Protocol Machine (DELAY_REQ) are described in the service definition and shown in Table 169.

Table 169 – Remote primitives issued or received by DELAY_REQ

Primitive	Source	Destination	Associated parameters	Functions
MAUType_Change_ind	IEEE Std 802.3	DELAY_REQ	PortID, MAUType, LINK_Status	—
LMPM_P_Data.ind	LMPM	DELAY_REQ	CREP, S_Port, Tstamp, DA, SA, A_SDU	The LMPM indicates a received delay response or delay follow up response.
LMPM_P_Data.cnf ()	LMPM	DELAY_REQ	CREP, D_Port, Tstamp, LMPM_status	—
LMPM_P_Data.req	DELAY_REQ	LMPM	CREP, D_Port, Tstamp, DA, SA, A_SDU	The DELAY_REQ issues a delay request.

4.4.4.2.1.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DELAY_REQ are described in the service definition and shown in Table 170.

Table 170 – Local primitives issued or received by DELAY_REQ

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.4.4.2.1.2 State transition diagram

The state transition diagram of the DELAY_REQ is shown in Figure 56.

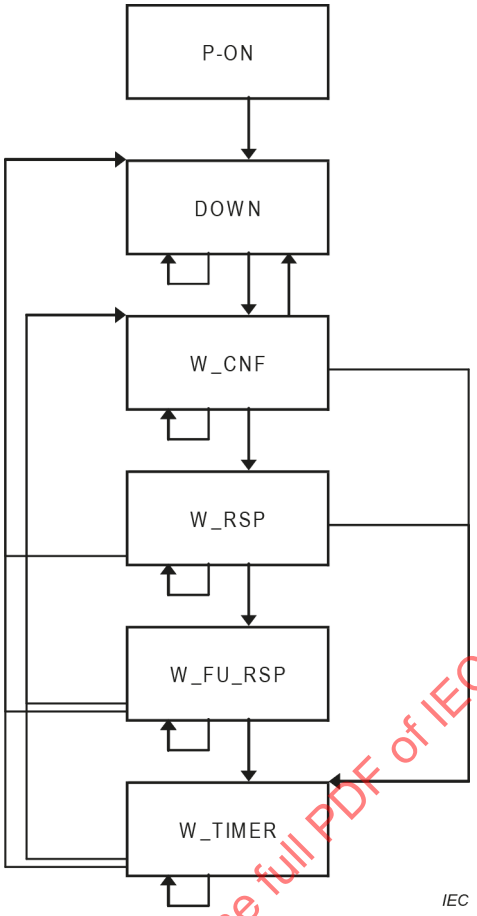


Figure 56 – State transition diagram of DELAY_REQ

States of the DELAY_REQ are:

P-ON	Initialization of local data.
DOWN	Link of port is down. The line delay measurement shall be initiated after link up by sending a delay request frame.
W_CNF	Wait for confirmation of delay request and store the transmit timestamp of the delay request message.
W_RSP	Wait for delay response frame from the DelayResponder.
W_FU_RSP	Wait for delay response follow up frame and calculate line delay.
W_TIMER	Wait for timeout to repeat the line delay measurement.

4.4.4.2.1.3 State machine description

The line delay measurement shall be initiated by each port and should be repeated in intervals of 8 s. The state machine which sends a delay request frame is called DelayRequestor.

4.4.4.2.1.4 DELAY_REQ state table

Table 171 contains the state table used by DELAY_REQ.

Table 171 – DELAY_REQ state table

#	Current State	Event /Condition =>Action	Next State
1	P-ON	=> Nrepeat := 0 SequenceID := 0 ErrorCounter := 0 RESET_RCF_PEER RESET_LINE_DELAY	DOWN
2	DOWN	**MAUType_Change_ind ()** /!LINK_OK => ignore	DOWN
3	DOWN	**MAUType_Change_ind ()** /LINK_OK => A_PDU := PTCP_DELAY_REQ_PDU StartTimer (DelayT, DELAY_REQ_SHORT_INTERVAL) StartTimer (LineT, TIMEOUT_LINE_DELAY) DelayReq_Req ()	W_CNF
4	W_CNF	**DelayReq_Cnf ()** /Status == !OK => SequenceID++	W_TIMER
5	W_CNF	**DelayReq_Cnf ()** /Status == OK => Tx_Tstamp_T1 := Tstamp	W_RSP
6	W_CNF	**MAUType_Change_ind ()** /!LINK_OK => Nrepeat := 0 ErrorCounter := 0 SequenceID++ StopTimer (DelayT) StopTimer (LineT) RESET_RCF_PEER RESET_LINE_DELAY	DOWN
7	W_CNF	**MAUType_Change_ind ()** /LINK_OK => ignore	W_CNF
8	W_CNF	**TimerExpired (DelayT)** => A_PDU := PTCP_DELAY_REQ_PDU SequenceID++ StartTimer (DelayT, DELAY_REQ_SHORT_INTERVAL) StartTimer (LineT, TIMEOUT_LINE_DELAY) DelayReq_Req ()	W_CNF
9	W_CNF	**TimerExpired (LineT)** /MAX_ERROR_COUNT > ErrorCounter => ErrorCounter++	W_TIMER
10	W_CNF	**TimerExpired (LineT)** /MAX_ERROR_COUNT <= ErrorCounter => ErrorCounter := 0 RESET_LINE_DELAY	W_TIMER

#	Current State	Event /Condition =>Action	Next State
11	W_RSP	**DelayResp_Ind ()** /!CHECK_DELAY_RSP_VALID => ignore	W_RSP
12	W_RSP	**DelayResp_Ind ()** /CHECK_DELAY_RSP_VALID && CHECK_DELAY_RSP_WITH_FU_RSP => Rx_Tstamp2_T2 := Rx_Tstamp1_T2 Rx_Tstamp1_T2 := PTCP_DELAY_RES_PDU.T2TimeStamp Delay := PTCP_DELAY_RES_PDU.Delay Rx_Tstamp_T4 := Tstamp if (Nrepeat > 0) CALC_RCF_PEER	W_FU_RSP
13	W_RSP	**DelayResp_Ind ()** /CHECK_DELAY_RSP_VALID && !CHECK_DELAY_RSP_WITH_FU_RSP && CHECK_DELAY_VALID => Rx_Tstamp2_T2 := Rx_Tstamp1_T2 Rx_Tstamp1_T2 := PTCP_DELAY_RES_PDU.T2TimeStamp Rx_Tstamp_T4 := Tstamp Delay := PTCP_DELAY_RES_PDU.Delay if (Nrepeat > 0) CALC_RCF_PEER CALC_LINE_DELAY Nrepeat++ ErrorCounter := 0	W_TIMER
14	W_RSP	**DelayResp_Ind ()** /CHECK_DELAY_RSP_VALID && !CHECK_DELAY_RSP_WITH_FU_RSP && !CHECK_DELAY_VALID => SET_DUMMY_LINE_DELAY StartTimer (DelayT, DELAY_REQ_SHORT_INTERVAL)	W_TIMER
15	W_RSP	**DelayFuResp_Ind ()** => ignore	W_RSP
16	W_RSP	**MAUType_Change_ind ()** /!LINK_OK => Nrepeat := 0 ErrorCounter := 0 SequenceID++ StopTimer (DelayT) StopTimer (LineT) RESET_RCF_PEER RESET_LINE_DELAY	DOWN
17	W_RSP	**MAUType_Change_ind ()** /LINK_OK => ignore	W_RSP
18	W_RSP	**TimerExpired (DelayT)** => SequenceID++ Nrepeat := 0 RESET_RCF_PEER StartTimer (DelayT, DELAY_REQ_LONG_INTERVAL)	W_TIMER
19	W_RSP	**TimerExpired (LineT)** /MAX_ERROR_COUNT > ErrorCounter => ErrorCounter++	W_TIMER

#	Current State	Event /Condition =>Action	Next State
20	W_RSP	**TimerExpired (LineT)** /MAX_ERROR_COUNT <= ErrorCounter => ErrorCounter := 0 RESET_LINE_DELAY	W_TIMER
21	W_FU_RSP	**DelayResp_Ind ()** /!CHECK_DELAY_RSP_VALID => ignore	W_FU_RSP
22	W_FU_RSP	**DelayResp_Ind ()** /CHECK_DELAY_RSP_VALID => Nrepeat := 0 StopTimer (DelayT) StartTimer (DelayT, DELAY_REQ_LONG_INTERVAL) RESET_LINE_DELAY	W_TIMER
23	W_FU_RSP	**DelayFuResp_Ind ()** /!CHECK_DELAY_FU_RSP_VALID => ignore	W_FU_RSP
24	W_FU_RSP	**DelayFuResp_Ind ()** /CHECK_DELAY_FU_RSP_VALID => Delay := Delay + PTCP_DELAY_FURES_PDU.Delay if (Nrepeat > 0) CALC_LINE_DELAY Nrepeat++ ErrorCounter := 0	W_TIMER
25	W_FU_RSP	**MAUType_Change_ind ()** /!LINK_OK => Nrepeat := 0 ErrorCounter := 0 SequenceID++ StopTimer (DelayT) StopTimer (LineT) RESET_RCF_PEER RESET_LINE_DELAY	DOWN
26	W_FU_RSP	**MAUType_Change_ind ()** /LINK_OK => ignore	W_FU_RSP
27	W_FU_RSP	**TimerExpired (DelayT)** => A_PDU := PTCP_DELAY_REQ_PDU SequenceID++ StartTimer (DelayT, DELAY_REQ_SHORT_INTERVAL) StartTimer (LineT, TIMEOUT_LINE_DELAY) DelayReq_Req ()	W_CNF
28	W_FU_RSP	**TimerExpired (LineT)** /MAX_ERROR_COUNT > ErrorCounter => ErrorCounter++	W_TIMER
29	W_FU_RSP	**TimerExpired (LineT)** /MAX_ERROR_COUNT <= ErrorCounter => ErrorCounter := 0 RESET_LINE_DELAY	W_TIMER

#	Current State	Event /Condition =>Action	Next State
30	W_TIMER	**DelayResp_Ind ()** => ignore	W_TIMER
31	W_TIMER	**DelayFuResp_Ind ()** => ignore	W_TIMER
32	W_TIMER	**MAUType_Change_ind ()** /!LINK_OK => Nrepeat := 0 ErrorCounter := 0 SequenceID++ StopTimer (DelayT) StopTimer (LineT) RESET_RCF_PEER RESET_LINE_DELAY	DOWN
33	W_TIMER	**MAUType_Change_ind ()** /LINK_OK => ignore	W_TIMER
34	W_TIMER	**TimerExpired (DelayT)** /Nrepeat < MAX_DELAY_REQ_REPEAT => A_PDU := PTCP_DELAY_REQ_PDU SequenceID++ StartTimer (DelayT, DELAY_REQ_SHORT_INTERVAL) StartTimer (LineT, TIMEOUT_LINE_DELAY) DelayReq_Req ()	W_CNF
35	W_TIMER	**TimerExpired (DelayT)** /Nrepeat >= MAX_DELAY_REQ_REPEAT => Nrepeat := 0 RESET_RCF_PEER StartTimer (DelayT, DELAY_REQ_LONG_INTERVAL)	W_TIMER
36	W_TIMER	**TimerExpired (LineT)** => ignore	W_TIMER

4.4.4.2.1.5 Functions, Macros, Timers and Variables

Table 172 contains the functions, macros, timers and variables used by the DELAY_REQ and their arguments and their descriptions.

Table 172 – Functions, macros, timers and variables used by the DELAY_REQ

Name	Type	Function/Meaning
CALC_LINE_DELAY	Function	Line delay for the arithmetical average value of at least the last 7 delay measurements.
CALC_RCF_PEER	Function	Calculation of the frequency deviation to the neighbor.
CHECK_DELAY_RSP_VALID	Function	Check whether the delay response is valid according to the coding rules.
CHECK_DELAY_RSP_WITH_FU_RSP	Function	Check if delay response type is with or without delay response follow up frame. TRUE := Wait for the delay response follow up FALSE := Without delay response follow up
CHECK_DELAY_VALID	Function	Check parameter of PTCP_DELAY_RSP_PDU. TRUE := The line delay measurement is supported. FALSE := “Special case”; the connected node does not support line delay measurement but wants to receive the sync messages. This case is signaled by PTCP_DELAY_RSP_PDU.DelayNs := 0x00
LINK_OK	Function	Check link MAUType supports FULL_DUPLEX LINK_Status == Up // LINK_Status == UP_CLOSED
RESET_LINE_DELAY	Function	Set line delay to zero.
RESET_RCF_PEER	Function	Set RCF_PEER to 1
SET_DUMMY_LINE_DELAY	Function	Set line delay internally to "DummyMode" and CableDelay to "Unknown". In this case neither the LineDelay nor the CableDelay is known. No valid Sync frame could be received from this port due to the unknown LineDelay. May be used to connect end stations without hardware support for synchronization.
StartTimer	Function	This local function is used to start or restart a timer.
StopTimer	Function	This local function is used to stop a timer.
TimerExpired	Function	This local function is issued if a timer expires.
DelayFuResp_Ind (S_Port, Tstamp, PTCP_DELAY_FU_RSP_PDU)	Macro	Receive PTCP-PDU according to PTCP_DELAY_FU_RSP_PDU. LMPM_P_Data.ind (CREP, D_Port, Tstamp, DA, SA, A_SDU) Assignments: PTCP_DELAY_FU_RSP_PDU := A_SDU without LT and FrameID
DelayReq_Cnf (D_Port, Tstamp, Status)	Macro	LMPM_P_Data.cnf (CREP, D_Port, Tstamp, LMPM_status) Assignments: Status := LMPM_status

Name	Type	Function/Meaning
DelayReq_Req (Port, PTCP_DELAY_REQ_PDU)	Macro	Create PTCP-PDU according to PTCP_DELAY_REQ_PDU. Assignments: D_Port := Port DA :=PTCP_MulticastMACadd for PTCP_DELAY_REQ_PDU SA := local port address PTCP_DELAY_REQ_PDU.SequenceID := SequenceID PTCP_DELAY_REQ_PDU.PortMACAddress := local port address A_SDU := LT, FrameID, PTCP_DELAY_REQ_PDU LMPM_P_Data.req (CREP, D_Port, Tstamp, DA, SA, A_SDU)
DelayResp_Ind (S_Port, Tstamp, PTCP_DELAY_RSP_PDU)	Macro	Receive PTCP-PDU according to PTCP_DELAY_RSP_PDU. LMPM_P_Data.ind (CREP, S_Port, Tstamp, DA, SA, A_SDU) Assignments: PTCP_DELAY_RSP_PDU := A_SDU without LT and FrameID
DelayT	Timer	This local timer is used to control the start of a line delay measurement.
LineT	Timer	This local timer is used to monitor the timely reception of a DelayResponse indication.
DELAY_REQ_LONG_INTERVAL	Variable	This local variable contains the time of the pause of line delay measurement. Default value: 8 s
DELAY_REQ_SHORT_INTERVAL	Variable	This local variable contains the time between two line delay measurements. Default value: 200 ms
ErrorCounter	Variable	This local variable contains the actual error count.
MAX_DELAY_REQ_REPEAT	Variable	This local variable contains the number of line delay measurements between a pause. Default value: 5
MAX_ERROR_COUNT	Variable	This local variable contains the number of allowed erroneous line delay measurements before the line delay is reset. Default value: 3
Rx_Tstamp_T4	Variable	This local variable contains the receive timestamp of the delay response frame.
Rx_Tstamp1_T2	Variable	This local variable contains a receive timestamp of the delay request frame by the delay responder.
Rx_Tstamp2_T2	Variable	This local variable contains a receive timestamp of the delay request frame by the delay responder.
SequenceID	Variable	This local variable contains the sequence number of the delay request message. It shall be incremented by one with every delay measurement.
TIMEOUT_LINE_DELAY	Variable	This local variable contains the timeout for the line delay measurement response. Default value: 100 ms
Tx_Tstamp_T1	Variable	This local variable contains the transmit timestamp of the delay request frame.

4.4.4.2.2 Line Delay Response Protocol Machine

4.4.4.2.2.1 Primitive definitions

4.4.4.2.2.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Line Delay Response Protocol Machine (DELAY_RSP) are described in the service definition and shown in Table 173.

Table 173 – Remote primitives issued or received by DELAY_RSP

Primitive	Source	Destination	Associated parameters	Functions
MAUType_Change_ind	IEEE Std 802.3	DELAY_RSP	PortID, MAUType, LINK_Status	—
LMPM_P_Data.ind	LMPM	DELAY_RSP	CREP, S_Port, Tstamp, DA, SA, A_SDU	—
LMPM_P_Data.cnf ()	LMPM	DELAY_RSP	CREP, D_Port, Tstamp, LMPM_status	—
LMPM_P_Data.req	DELAY_RSP	LMPM	CREP, D_Port, Tstamp, DA, SA, A_SDU	—

4.4.4.2.2.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DELAY_RSP are described in the service definition and shown in Table 174.

Table 174 – Local primitives issued or received by DELAY_RSP

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.4.4.2.2.2 State transition diagram

The state transition diagram of the DELAY_RSP is shown in Figure 57.

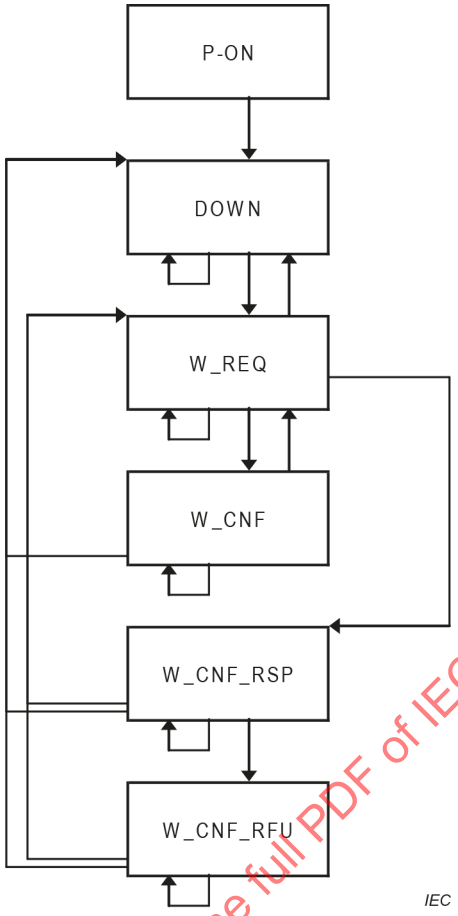


Figure 57 – State transition diagram of DELAY_RSP

States of the DELAY_RSP are:

P-ON	Data initialization.
DOWN	Link of port is down. Wait for link up.
W_REQ	If a delay request message is received the receive time stamp will be stored and the DelayResponder sends a delay response message.
W_CNF	Wait for the confirmation of the delay response request.
W_CNF_RSP	Wait for the confirmation of the delay response message, calculate the delay response time and send a delay response follow up message.
W_CNF_RFU	Wait for the confirmation of the delay response follow up message.

4.4.4.2.2.3 State machine description

The DelayResponder state machine handles the responder functionality of the line delay measurement. It receives the line delay measurement request and responds with a line delay measurement response.

The DelayResponder shall respond with the DelayResponse without FollowUp, if supported. Otherwise it shall use the DelayResponse with FollowUp.

Each delay request message shall be immediately responded to with a delay response message.

4.4.4.2.2.4 DELAY_RSP state table

Table 175 contains the state table used by DELAY_RSP.

Table 175 – DELAY_RSP state table

#	Current State	Event /Condition =>Action	Next State
1	P-ON	=>	DOWN
2	DOWN	**MAUType_Change_ind ()** /! LINK_OK => ignore	DOWN
3	DOWN	**MAUType_Change_ind ()** /LINK_OK => ignore	W_REQ
4	W_REQ	**DelayReq_Ind ()** /FU_RES => store PTCP_DELAY_REQ_PDU A_PDU := PTCP_DELAY_RES_PDU Rx_Tstamp_T2 := Tstamp PTCP_DELAY_RES_PDU.T2TimeStamp := Rx_Tstamp_T2 DelayResp_Req ()	W_CNF_RSP
5	W_REQ	**DelayReq_Ind ()** /!FU_RES => store PTCP_DELAY_REQ_PDU A_PDU := PTCP_DELAY_RES_PDU Rx_Tstamp_T2 := Tstamp PTCP_DELAY_RES_PDU.T2TimeStamp := Rx_Tstamp_T2 DelayResp_Req ()	W_CNF
6	W_REQ	**DelayReq_Ind ()** /!FU_TIMESTAMP => store PTCP_DELAY_REQ_PDU A_PDU := PTCP_DELAY_RES_PDU PTCP_DELAY_RES_PDU.Delay := 0 DelayResp_Req ()	W_CNF
7	W_REQ	**MAUType_Change_ind ()** /! LINK_OK => ignore	DOWN
8	W_REQ	**MAUType_Change_ind ()** /LINK_OK => ignore	W_REQ
9	W_CNF	**DelayResp_Cnf ()** => ignore	W_REQ
10	W_CNF	**MAUType_Change_ind ()** /! LINK_OK => ignore	DOWN
11	W_CNF	**MAUType_Change_ind ()** /LINK_OK => ignore	W_CNF

#	Current State	Event /Condition =>Action	Next State
12	W_CNF_RSP	**DelayResp_Cnf ()** /Status != OK => ignore	W_REQ
13	W_CNF_RSP	**DelayResp_Cnf ()** /Status == OK => Tx_Tstamp_T3 := Tstamp A_PDU := PTCP_DELAY_FU_RES_PDU PTCP_DELAY_FU_RES_PDU.Delay := CALC_RESIDENTIAL_TIME DelayFuResp_Req ()	W_CNF_RFU
14	W_CNF_RSP	**MAUType_Change_ind ()** /! LINK_OK => ignore	DOWN
15	W_CNF_RSP	**MAUType_Change_ind ()** /LINK_OK => ignore	W_CNF_RSP
16	W_CNF_RFU	**DelayFuResp_Cnf ()** => ignore	W_REQ
17	W_CNF_RFU	**MAUType_Change_ind ()** /! LINK_OK => ignore	DOWN
18	W_CNF_RFU	**MAUType_Change_ind ()** /LINK_OK => ignore	W_CNF_RFU

4.4.4.2.2.5 Functions, Macros, Timers and Variables

Table 176 contains the functions, macros, timers and variables used by the DELAY_RSP and their arguments and their descriptions.

Table 176 – Functions, Macros, Timers and Variables used by the DELAY_RSP

Name	Type	Function/Meaning
DelayResp_Req (Port, PTCP_DELAY_RSP_PDU)	Macro	Create PTCP-PDU according to PTCP_DELAY_RSP_PDU Assignments: D_Port := Port DA := Multicast-MAC-Address for PTCP_DELAY_RSP_PDU SA := local port address PTCP_DELAY_RSP_PDU.SequenceID := PTCP_DELAY_REQ_PDU.SequenceID PTCP_DELAY_RSP_PDU.PortMACAddress := PTCP_DELAY_REQ_PDU.PortMACAddress PTCP_DELAY_REQ_PDU.T2PortRxDelay := T2_PortRxDelay PTCP_DELAY_REQ_PDU.T3PortTxDelay := T3_PortTxDelay PTCP_DELAY_REQ_PDU.T2TimeStamp := T2_TimeStamp A_SDU := LT, FrameID, PTCP_DELAY_RSP_PDU LMPM_P_Data.req (CREP, D_Port, Tstamp, DA, SA, A_SDU)
DelayFuResp_Req (Port, ResTime, PTCP_DELAY_FU_RSP_PDU)	Macro	Create PTCP-PDU according to PTCP_DELAY_FU_RSP_PDU Assignments: D_Port := Port DA := Multicast-MAC-Address for PTCP_DELAY_FU_RSP_PDU SA := local source address PTCP_Delay := ResTime A_SDU := LT, FrameID, PTCP_DELAY_FU_RSP_PDU LMPM_P_Data.req (CREP, D_Port, Tstamp, DA, SA, A_SDU)
DelayResp_Cnf (D_Port, Tstamp, Status)	Macro	LMPM_P_Data.cnf (CREP, D_Port, Tstamp, LMPM_status) Assignments: Status := LMPM_status
DelayFuResp_Cnf (D_Port, Tstamp, Status)	Macro	LMPM_P_Data.cnf (CREP, D_Port, Tstamp, LMPM_Status) Assignments: Status := LMPM_status
DelayReq_Ind (S_Port, Tstamp, PTCP_DELAY_REQ_PDU)	Macro	Receive PTCP-PDU according to PTCP_DELAY_REQ_PDU LMPM_P_Data.ind (CREP, S_Port, Tstamp, DA, SA, A_SDU) Assignments: PTCP_DELAY_REQ_PDU := A_SDU without LT and FrameID
Rx_Tstamp_T2	Variable	This local variable contains the receive timestamp of the delay request message.
Tx_Tstamp_T3	Variable	This local variable contains the transmit timestamp of the delay response message.
LINK_OK	Macro	Check whether the MAUType supports FULL_DUPLEX and the LINK_Status is UP or BLOCKED.
CALC_RESIDENTIAL_TIME	Macro	Calculate the residential time ResTime := (Tx_Stamp_T3 – Rx_Stamp_T2)

4.4.4.3 Overview of state tables for master and slave

Figure 58 shows an overview of the interaction between the PTCP state tables. For devices without PTCP support or for PTCP slaves without an active PTCP master, the reference time sink is generated by local means.

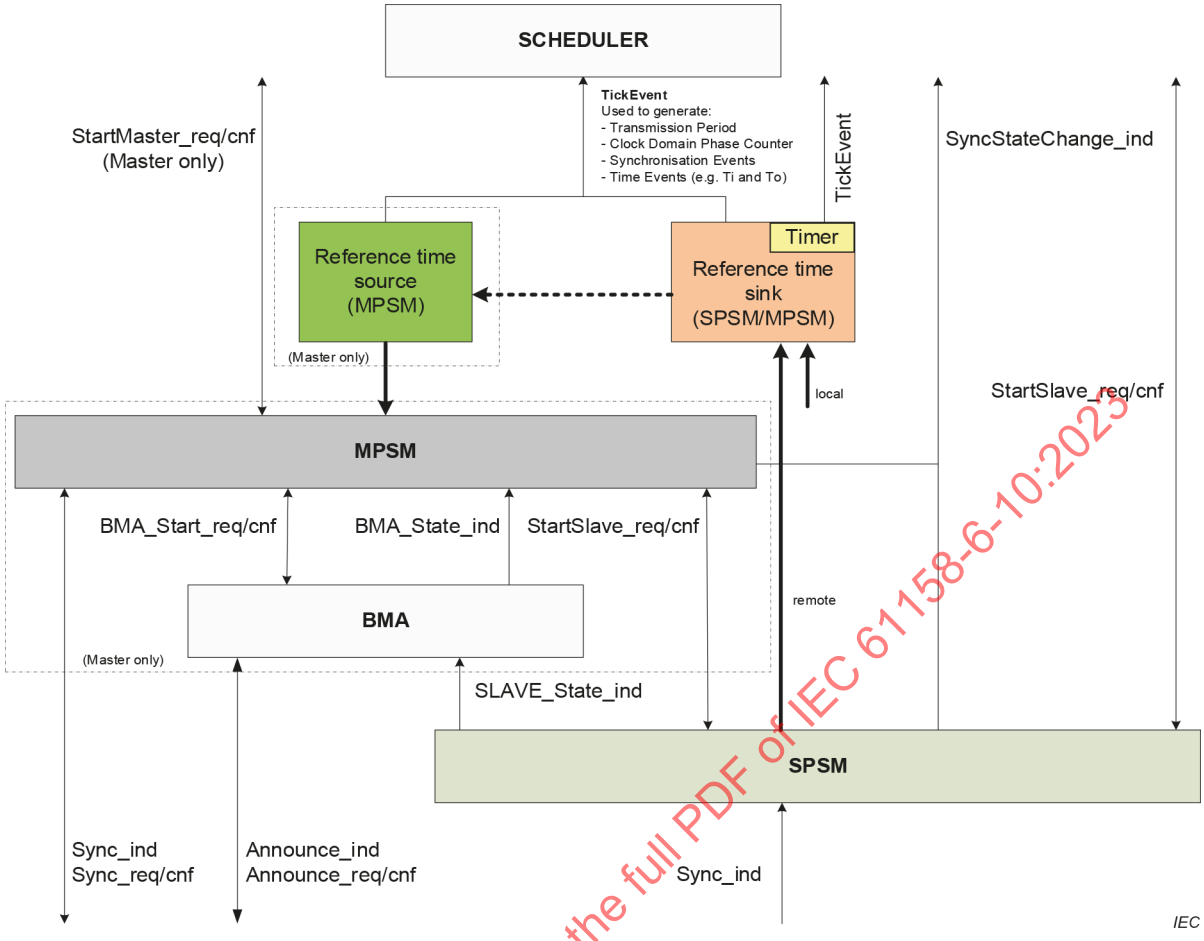


Figure 58 – Overview of PTCP

For the use of PTCP in conjunction with wireless the principles of IEEE Std 802.1AS should be applied.

4.4.4.4 Best-Master-Algorithm

4.4.4.4.1 Best-Master-Algorithm Protocol Machine

4.4.4.4.1.1 Primitive definitions

####### 4.4.4.4.1.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Best-Master-Algorithm Protocol Machine (SYN_BMA) are described in the service definition and shown in Table 177.

Table 177 – Remote primitives issued or received by SYN_BMA

Primitive	Source	Destination	Associated parameters	Functions
LMPM_A_Data.req	BMA	LMPM	CREP, D_Port, Tstamp, DA, SA, A_SDU	—
LMPM_A_Data.cnf	LMPM	BMA	CREP, D_Port, Tstamp, LMPM_status	—
LMPM_A_Data.ind	LMPM	BMA	CREP, S_Port, Tstamp, DA, SA, A_SDU	—

4.4.4.4.1.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by SYN_BMA are described in the service definition and shown in Table 178.

Table 178 – Local primitives issued or received by SYN_BMA

Primitive	Source	Destination	Associated parameters	Functions
BMA_Start_req	MPSM	BMA	—	Activate best master function
BMA_Start_cnf (+/-)	BMA	MPSM	—	Confirmation for best master function start
BMA_Stop_req	MPSM	BMA	—	Deactivate best master function
BMA_Stop_cnf (+/-)	BMA	MPSM	—	Confirmation for stop best master function
BMA_State_ind	BMA	MPSM	Role	During the startup period the BMA elects the best sync master. Allowed value for Role: - MASTER - SLAVE
SLAVE_State_ind	SPSM	BMA	State	Indication for master lost. The event is generated by the SLAVE. - MASTER_LOST

Primitive	Source	Destination	Associated parameters	Functions
Announce_req	BMA	LMPM	PTCP_ANNOUNCE_PDU	Create PTCP-PDU according to PTCP_ANNOUNCE_PDU Assignments: D_Port := AUTO DA :=PTCP_MulticastMACadd for PTCP_ANNOUNCE_PDU SA := local source address PTCP_ANNOUNCE_PDU.SequenceID := SequenceID PTCP_ANNOUNCE_PDU.DomainUUID := DomainUUID PTCP_ANNOUNCE_PDU.MasterSourceAddress := local master address PTCP_ANNOUNCE_PDU.MasterPriority1 := Priority PTCP_ANNOUNCE_PDU.ClockClass := Class PTCP_ANNOUNCE_PDU.ClockAccuracy := Accuracy PTCP_ANNOUNCE_PDU.ClockVariance := Variance PTCP_ANNOUNCE_PDU.MasterPriority2 := 0xFF A_SDU := LT, FrameID, PTCP_ANNOUNCE_PDU LMPM_A_Data.req (CREP, D_Port, Tstamp, DA, SA, A_SDU)
Announce_cnf (+/-)	LMPM	BMA	State	LMPM_A_Data.cnf (CREP, D_Port, Tstamp, LMPM_status) Assignments: Status := LMPM_status
Announce_ind	LMPM	BMA	PTCP_ANNOUNCE_PDU	Receive PTCP-PDU according to PTCP_SYNC_PDU LMPM_A_Data.ind (CREP, S_Port, Tstamp, DA, SA, A_SDU) Assignments: PTCP_SYNC_PDU := A_SDU without LT and FrameID

4.4.4.4.1.2 State transition diagram

The state transition diagram of the SYN_BMA is shown in Figure 59:

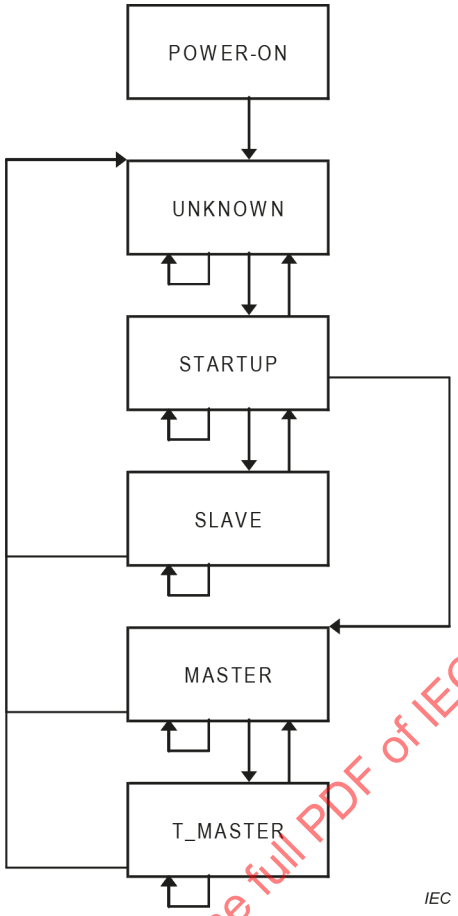


Figure 59 – State transition diagram of SYN_BMA

States of the SYN_BMA are:

POWER-ON	Data initialization
UNKNOWN	The synchronization role is unknown. A role shall be set via PTCP services.
STARTUP	The role set via PTCP services. The potential master starts sending announce messages.
SLAVE	The potential master in state SLAVE shall be synchronized by using the synchronization messages from the active master in the sync domain.
MASTER	At this state the device is, due to its synchronization attributes, elected best master.
T_MASTER	This is a transition state which indicates the receipt of announce messages from another potential master in the sync domain.

4.4.4.4.1.3 State machine description

The BMA state machine shall elect the best master from all announced master-capable devices during the startup period. These decisions are made locally by every master. An instance of the state machine exists for every PTCP SyncID.

Due to the fact that not every device is able to be master or should be master, the behavior of the state machine depends on its role. The role is set by PTCP services. Roles are master, secondary master and slave. A master also needs slave functionality.

Whether a device with the role master becomes master or secondary master will be decided during the startup period by the best master algorithm. Every potential master shall send announce messages to take part in master and secondary master election.

The election of the master will be done in two steps. With the first received announce the potential master will be stored in a master list. With the next announce the SYN_BMA elects this master, if it is the best known master.

During master competition all potential masters will be collected in a master list. After election they drop out via aging.

If the decision to be master is made, the best master starts sending sync messages. The remaining potential masters stop to send announce messages and shall be slaves. If the active master fails, the potential masters restart sending sync and announce messages.

4.4.4.4.1.4 SYN_BMA state table

Table 179 contains the state table used by SYN_BMA.

Table 179 – SYN_BMA state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> init MasterList SequenceID := 0 BestMaster := FALSE ACTIVE_MASTER_FLAG := FALSE AGING_MASTER_LIST StartTimer (MasterListAgingTimer, MasterAgingTime)	UNKNOWN
2	UNKNOWN	**BMA_Start_req ()** => Status := OK A_PDU := PTCP_ANNOUNCE_PDU StartTimer (AnnTimer, MasterAnnounceTime) StartTimer (StartupTimer, MasterStartupTime) BMA_Start_cnf (+) for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Announce_req (i, PTCP_ANNOUNCE_PDU)	STARTUP
3	UNKNOWN	**BMA_Stop_req ()** => StopTimer (AnnTimer) Status := OK BMA_Stop_cnf (+)	UNKNOWN
4	UNKNOWN	**TimerExpired (MasterListAgingTimer)** => AGING_MASTER_LIST StartTimer (MasterListAgingTimer, MasterAgingTime)	UNKNOWN
5	STARTUP	**BMA_Start_req ()** => Status := WRONG_SEQUENCE BMA_Start_cnf (-)	STARTUP
6	STARTUP	**BMA_Stop_req ()** => StopTimer (AnnTimer) StopTimer (StartupTimer) Status := OK BMA_Stop_cnf (+)	UNKNOWN

#	Current State	Event /Condition =>Action	Next State
7	STARTUP	**Announce_cnf ()** => ignore	STARTUP
8	STARTUP	**SLAVE_State_ind (State)** => ignore	STARTUP
9	STARTUP	**Announce_ind ()** /!VALID_SYNC_MASTER => ignore	STARTUP
10	STARTUP	**Announce_ind ()** /VALID_SYNC_MASTER && !CHECK_IN_MASTER_LIST => ADD_TO_MASTER_LIST	STARTUP
11	STARTUP	**Announce_ind ()** /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST => UPDATE_MASTER_ENTRY	STARTUP
12	STARTUP	**TimerExpired (AnnTimer)** /!LOCAL_IS_BEST_MASTER => BestMaster := FALSE StopTimer (StartupTimer) StartTimer (AnnTimer, MasterAnnounceTime) BMA_State_ind (SLAVE)	SLAVE
13	STARTUP	**TimerExpired (AnnTimer)** /LOCAL_IS_BEST_MASTER => BestMaster := TRUE SequenceID++ A_PDU := PTCP_ANNOUNCE_PDU StartTimer (AnnTimer, MasterAnnounceTime) BMA_State_ind (MASTER) for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Announce_req (i, PTCP_ANNOUNCE_PDU)	STARTUP
14	STARTUP	**TimerExpired (StartupTimer)** => ACTIVE_MASTER_FLAG := TRUE	MASTER
15	STARTUP	**TimerExpired (MasterListAgingTimer)** => AGING_MASTER_LIST StartTimer (MasterListAgingTimer, MasterAgingTime)	STARTUP
16	SLAVE	**BMA_Start_req ()** => Status := WRONG_SEQUENCE BMA_Start_cnf (-)	SLAVE
17	SLAVE	**BMA_Stop_req ()** => StopTimer (AnnTimer) Status := OK BMA_Stop_cnf (+)	UNKNOWN
18	SLAVE	**Announce_cnf ()** => ignore	SLAVE
19	SLAVE	**SLAVE_State_ind (State)** /State != MASTER_LOST => ignore	SLAVE

#	Current State	Event /Condition =>Action	Next State
20	SLAVE	**SLAVE_State_ind (State)** /State == MASTER_LOST && !BestMaster => ignore	SLAVE
21	SLAVE	**SLAVE_State_ind (State)** /State == MASTER_LOST && BestMaster => SequenceID++ A_PDU:= PTCP_ANNOUNCE_PDU StartTimer (StartupTimer, MasterStartupTime) BMA_State_ind (MASTER) for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Announce_req (i, PTCP_ANNOUNCE_PDU)	STARTUP
22	SLAVE	**Announce_ind ()** /!VALID_SYNC_MASTER => ignore	SLAVE
23	SLAVE	**Announce_ind ()** /VALID_SYNC_MASTER && !CHECK_IN_MASTER_LIST => ADD_TO_MASTER_LIST	SLAVE
24	SLAVE	**Announce_ind ()** /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST => UPDATE_MASTER_ENTRY	SLAVE
25	SLAVE	**TimerExpired (AnnTimer)** /!LOCAL_IS_BEST_MASTER => BestMaster := FALSE StartTimer (AnnTimer, MasterAnnounceTime)	SLAVE
26	SLAVE	**TimerExpired (AnnTimer)** /LOCAL_IS_BEST_MASTER => BestMaster := TRUE StartTimer (AnnTimer, MasterAnnounceTime)	SLAVE
27	SLAVE	**TimerExpired (MasterListAgingTimer)** => AGING_MASTER_LIST StartTimer (MasterListAgingTimer, MasterAgingTime)	SLAVE
28	MASTER	**BMA_Start_req ()** => Status := WRONG_SEQUENCE BMA_Start_cnf (-)	MASTER
29	MASTER	**BMA_Stop_req ()** => StopTimer (AnnTimer) Status := OK BMA_Stop_cnf (+)	UNKNOWN
30	MASTER	**Announce_cnf ()** => ignore	MASTER
31	MASTER	**SLAVE_State_ind (State)** => ignore	MASTER

#	Current State	Event /Condition =>Action	Next State
32	MASTER	**Announce_ind ()** /!VALID_SYNC_MASTER => ignore	MASTER
33	MASTER	**Announce_ind ()** /VALID_SYNC_MASTER && !CHECK_IN_MASTER_LIST => ADD_TO_MASTER_LIST	MASTER
34	MASTER	**Announce_ind ()** /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST => UPDATE_MASTER_ENTRY	MASTER
35	MASTER	**TimerExpired (AnnTimer)** /!MASTER_LIST_EMPTY => SequenceID++ A_PDU := PTCP_ANNOUNCE_PDU StartTimer (AnnTimer, MasterAnnounceTime) StartTimer (StartupTimer, MasterStartupTime) for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Announce_req (i, PTCP_ANNOUNCE_PDU)	T_MASTER
36	MASTER	**TimerExpired (AnnTimer)** /MASTER_LIST_EMPTY => StartTimer (AnnTimer, MasterAnnounceTime)	MASTER
37	MASTER	**TimerExpired (MasterListAgingTimer)** => AGING_MASTER_LIST StartTimer (MasterListAgingTimer, MasterAgingTime)	MASTER
38	T_MASTER	**BMA_Start_req ()** => Status := WRONG_SEQUENCE BMA_Start_cnf (-)	T_MASTER
39	T_MASTER	**BMA_Stop_req ()** => StopTimer (AnnTimer) Status := OK BMA_Stop_cnf (+)	UNKNOWN
40	T_MASTER	**Announce_cnf ()** => ignore	T_MASTER
41	T_MASTER	**SLAVE_State_ind (State)** => ignore	T_MASTER
42	T_MASTER	**Announce_ind ()** /!VALID_SYNC_MASTER => ignore	T_MASTER
43	T_MASTER	**Announce_ind ()** /VALID_SYNC_MASTER && !CHECK_IN_MASTER_LIST => ADD_TO_MASTER_LIST StartTimer (StartupTimer, MasterStartupTime)	T_MASTER
44	T_MASTER	**Announce_ind ()** /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST => UPDATE_MASTER_ENTRY StartTimer (StartupTimer, MasterStartupTime)	T_MASTER

#	Current State	Event /Condition =>Action	Next State
45	T_MASTER	**TimerExpired (AnnTimer)** /!MASTER_LIST_EMPTY => SequenceID++ A_PDU := PTCP_ANNOUNCE_PDU StartTimer (AnnTimer, MasterAnnounceTime) StartTimer (StartupTimer, MasterStartupTime) for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Announce_req (i, PTCP_ANNOUNCE_PDU)	T_MASTER
46	T_MASTER	**TimerExpired (AnnTimer)** /MASTER_LIST_EMPTY => SequenceID++ A_PDU := PTCP_ANNOUNCE_PDU StartTimer (AnnTimer, MasterAnnounceTime) for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Announce_req (i, PTCP_ANNOUNCE_PDU)	T_MASTER
47	T_MASTER	**TimerExpired (StartupTimer)** => ignore	MASTER
48	T_MASTER	**TimerExpired (MasterListAgingTimer)** => AGING_MASTER_LIST StartTimer (MasterListAgingTimer, MasterAgingTime)	T_MASTER

4.4.4.4.1.5 Functions, Macros, Timers and Variables

Table 180 contains the functions, macros, timers and variables used by the SYN_BMA and their arguments and their descriptions.

Table 180 – Functions, Macros, Timers and Variables used by the SYN_BMA

Name	Type	Function/Meaning
StartTimer	Function	This local function starts or restarts a timer.
StopTimer	Function	This local function stops a timer.
TimerExpired	Function	This local function is issued if a timer expires.
ADD_TO_MASTER_LIST	Macro	Insert sync master in remote sync master list. sm_sa := PTCP_ANNOUNCE_PDU.MasterSourceAddress if (MasterList.Num_entry < MAX_CNT_REMOTE_MASTER) MasterList.Insert(sm_sa) MasterList.Entry{sm_sa}.Priority1 := PTCP_ANNOUNCE_PDU.Priority1 MasterList.Entry{sm_sa}.Accuracy := PTCP_ANNOUNCE_PDU.Accuracy MasterList.Entry{sm_sa}.Variance := PTCP_ANNOUNCE_PDU.Variance MasterList.Entry{sm_sa}.Priority2 := PTCP_ANNOUNCE_PDU.Priority2 MasterList.Entry{sm_sa}.SA := sm_sa MasterList.Entry{sm_sa}.Receipt := 1 MasterList.Num_entry++
AGING_MASTER_LIST	Macro	Aging of entries in remote sync master list. for m := 0 to MasterList.Num_entry if (MasterList.Entry[m].Receipt == 0) MasterList.Num_entry-- MasterList.Remove(sm_sa)

Name	Type	Function/Meaning
Announce_cnf (+/-) (State)	Macro	LMPM_A_Data.cnf (CREP, D_Port, Tstamp, LMPM_status) Assignments: Status := LMPM_status
Announce_ind (PTCP_ANNOUNCE_PDU)	Macro	Receive PTCP-PDU according to PTCP_ANNOUNCE_PDU. LMPM_A_Data.ind (CREP, S_Port, Tstamp, DA, SA, A_SDU) Assignments: PTCP_ANNOUNCE_PDU := A_SDU without LT and FrameID
Announce_req (Port, PTCP_ANNOUNCE_PDU)	Macro	Create PTCP-PDU according to PTCP_ANNOUNCE_PDU. Assignments: D_Port := Port DA :=PTCP_MulticastMACadd for PTCP_ANNOUNCE_PDU SA := local source address PTCP_ANNOUNCE_PDU.SequenceID := SequenceID PTCP_ANNOUNCE_PDU.DomainUUID := DomainUUID PTCP_ANNOUNCE_PDU.MasterSourceAddress := local master address PTCP_ANNOUNCE_PDU.MasterPriority1 := Priority PTCP_ANNOUNCE_PDU.ClockClass := Class PTCP_ANNOUNCE_PDU.ClockAccuracy := Accuracy PTCP_ANNOUNCE_PDU.ClockVariance := Variance PTCP_ANNOUNCE_PDU.MasterPriority2 := 0xFF A_SDU := LT, FrameID, PTCP_ANNOUNCE_PDU LMPM_A_Data.req (CREP, D_Port, Tstamp, DA, SA, A_SDU)
CHECK_IN_MASTER_LIST	Macro	Master in PTCP_SubdomainUUID master list. sm_sa := PTCP_ANNOUNCE_PDU.MasterSourceAddress sm_sa in MasterList
init MasterList	Macro	Initialize data set for remote master. MasterList.Num_entry := 0
LOCAL_IS_BEST_MASTER	Macro	Execute comparison according 4.4.4.4.1.6 and return TRUE if the local master is found, else FALSE.
TRANSMIT_PORT_VALID	Macro	Table 195 shall be applied.
UPDATE_MASTER_ENTRY	Macro	Update entry in remote sync master list. MasterList.Entry{sm_sa}.Priority1 := PTCP_ANNOUNCE_PDU.Priority1 MasterList.Entry{sm_sa}.Accuracy := PTCP_ANNOUNCE_PDU.Accuracy MasterList.Entry{sm_sa}.Variance := PTCP_ANNOUNCE_PDU.Variance MasterList.Entry{sm_sa}.Priority2 := PTCP_ANNOUNCE_PDU.Priority2 MasterList.Entry{sm_sa}.Receipt += 1
VALID_SYNC_MASTER	Macro	Domain is supported. PTCP_ANNOUNCE_PDU.DomainUUID == DomainUUID
AnnTimer	Timer	This local timer is used to create the interval for the sending of announce frames.
MasterListAgingTimer	Timer	This local timer is used to create the interval for the aging of the entries in the master list.
StartupTimer	Timer	This local timer is used to create the interval for the checking of announce frames.
MasterAgingTime	Variable	Time for the aging of the master list. Default value: “2 x PTCP Takeover Timeout”
MasterAnnounceTime	Variable	An announce frame shall be sent every MasterAnnounceTime until MasterStartupTime. Default value: 100 ms
MasterStartupTime	Variable	Time for the receiving check of announce frames. Default value: 10 s

4.4.4.4.1.6 Finding the Best Sync Master

PTCP contains the support of multiple sync masters. The filtering of the stored sync masters to select the sync master to be used is done by the BMA using the key attributes SyncID and PTCP_SubdomainUUID, and the following checking rules:

NOTE 1 The PTCP_ClockClass is not used.

NOTE 2 Lower values take precedence.

1) Check whether a PTCP_MasterPriority1.Active == TRUE sync master exists

   The BMA checks whether an ACTIVE sync master exists. If another sync master is active, the checking master changes to the sync slave mode.

   If no ACTIVE sync master exists, continue with the next check.

NOTE 3 If the local sync master is active, it stays active until the application decides otherwise.

2) Find the sync master with the best PTCP_MasterPriority1.Level

   The BMA searches for the sync masters with the best value, taking its own value into account.

   If more than one sync master exists in residual list, then continue with the next check, else use the found one.

3) Find the sync master with the best PTCP_MasterPriority1.Priority

   The BMA searches for the sync masters with the best value in the residual list.

   If more than one sync master exists in residual list, then continue with the next check, else use the found one.

4) Find the sync master with the best PTCP_MasterPriority2

   The BMA searches for the sync masters with the best value in the residual list.

   If more than one sync master exists in residual list, then continue with the next check, else use the found one.

5) Find the sync master with the best PTCP_ClockVariance

   The BMA searches for the sync masters with the best value in the residual list.

   If more than one sync master exists in residual list, then continue with the next check, else use the found one.

6) Find the sync master with the best PTCP_ClockAccuracy

   The BMA searches for the sync masters with the best value in the residual list.

   If more than one sync master exists in residual list, then continue with the next check, else use the found one.

7) Find the sync master with the best PTCP_MasterSourceAddress

   The BMA searches for the sync masters with the lowest value (comparing octets zero to five) of the PTCP_MasterSourceAddress in the residual list.

   Use the found one.

NOTE 4 The PTCP_MasterSourceAddress is used as “tiebreaker”.

4.4.4.5 PTCP Master Protocol Machine

4.4.4.5.1 Primitive definitions

4.4.4.5.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by PTCP Master Protocol Machine (SYN_MPSM) are described in the service definition and shown in Table 181.

Table 181 – Remote primitives issued or received by SYN_MPSM

Primitive	Source	Destination	Associated parameters	Functions
LMPM_P_Data.req	MPSM	LMPM	CREP, D_Port, Tstamp, DA, SA, A_SDU	—
LMPM_P_Data.cnf	LMPM	MPSM	CREP, D_Port, Tstamp, LMPM_status	—
LMPM_P_Data.ind	LMPM	MPSM	CREP, S_Port, Tstamp, DA, SA, A_SDU	—

4.4.4.5.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by SYN_MPSM are described in the service definition and shown in Table 182.

Table 182 – Local primitives issued or received by SYN_MPSM

Primitive	Source	Destination	Associated parameters	Functions
StartMaster_req	PTCP ASE	MPSM	—	Activate master function
StartMaster_cnf	MPSM	PTCP ASE	Status	Confirmation of master start
StopMaster_req	PTCP ASE	MPSM	—	Deactivate master function
StopMaster_cnf	MPSM	PTCP ASE	Status	Confirmation of master stop
StartSlave_req	MPSM	SPSM	—	Activate slave function
StartSlave_cnf	SPSM	MPSM	—	Confirmation of start slave
StopSlave_req	MPSM	SPSM	—	Deactivate slave function
StopSlave_cnf	SPSM	MPSM	—	Confirmation of stop slave
BMA_Start_req	MPSM	BMA	—	Activate best master function
BMA_Start_cnf	BMA	MPSM	—	Confirmation of best master function start
BMA_Stop_req	MPSM	BMA	—	Deactivate best master function
BMA_Stop_cnf	BMA	MPSM	—	Confirmation of stop best master function
BMA_State_ind	BMA	MPSM	Role	During the startup period the BMA elects the best sync master. Allowed value for Role: - MASTER - SLAVE
SyncStateChange_ind	MSPM	FSPM	Status	Indication for events - SINGLE_MASTER - MULTIPLE_MASTER

4.4.4.5.2 State transition diagram

The state transition diagram of the SYN_MPSM is shown in Figure 60:

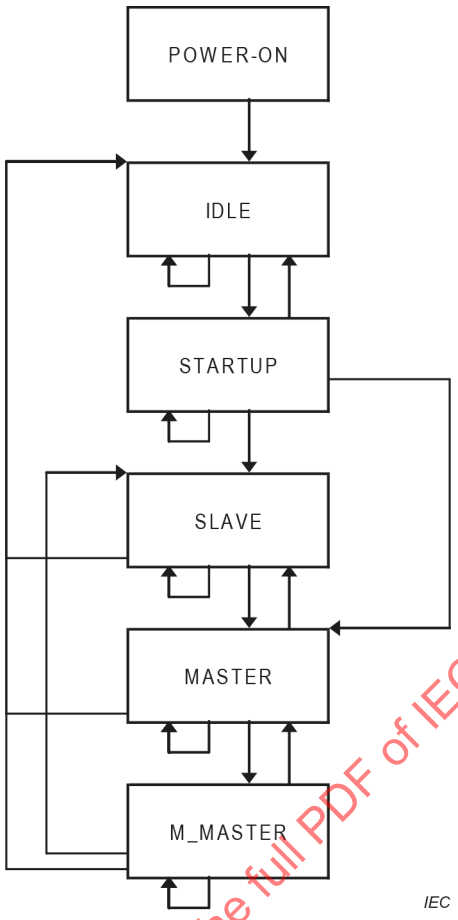


Figure 60 – State transition diagram of SYN_MPSM

States of the SYN_MPSM are:

POWER-ON	After startup the data management shall be initialized (POWER-ON) and the synchronization role shall be set (IDLE).
IDLE	The master role is set by PTCP services. After this the MPSM changes from state IDLE to state STARTUP and starts the BMA state machine.
STARTUP	During the startup period the MPSM is in state STARTUP.
SLAVE	The master is synchronized by sync messages from another master in the sync domain.
MASTER	This is the active primary master. Therefore it sends synchronization messages.
M_MASTER	This state handles the multiple master issues.

4.4.4.5.3 State machine description

The master protocol state machine controls the transition between the roles master and slave. The transitions are done locally by every master for every supported PTCP SyncID and the PTCP_SubdomainUUID assigned by engineering. Therefore an instance of the state machine exists for every SyncID with the key attribute SubdomainUUID. To achieve these transitions the MPSM shall initiate synchronization and announce messages. Announce messages shall be sent by the SYN_BMA to advertise master capabilities and ensure that only one master is active.

Multiple active primary masters shall be prevented by the state STARTUP and the best master algorithm. Therefore the decision to go from state STARTUP to state MASTER (primary sync master) will be made by the SYN_BMA with extensive knowledge of concurrent masters.

If an active master (with fewer capabilities) already exists and after the startup period a new master emerges, the new master needs to synchronize to the active master. For synchronization the MPSM changes to state SLAVE.

4.4.4.5.4 SYN_MPSM state table

Table 183 contains the state table used by SYN_MPSM.

Table 183 – SYN_MPSM state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> SequenceID := 0 MultipleMaster := FALSE	IDLE
2	IDLE	**StartMaster_req ()** => Status := OK BMA_Start_req () StartMaster_cnf (Status)	STARTUP
3	IDLE	**StopMaster_req ()** => Status := OK StopMaster_cnf (Status)	IDLE
4	IDLE	**Sync_cnf ()** => ignore	IDLE
5	IDLE	**BMA_Stop_cnf ()** => ignore	IDLE
6	STARTUP	**StartMaster_req ()** => Status := WRONG_SEQUENCE StartMaster_cnf (Status)	STARTUP
7	STARTUP	**StopMaster_req ()** => Status := OK StopMaster_cnf (Status) BMA_Stop_req ()	IDLE
8	STARTUP	**Sync_cnf ()** => ignore	STARTUP
9	STARTUP	**BMA_State_ind (State)** /State == SLAVE => Status := OK StartSlave_req ()	SLAVE
10	STARTUP	**BMA_State_ind (State)** /State == MASTER => State := SINGLE_MASTER SequenceID++ StartTimer (SyncTimer, SyncInterval) StartTimer (TimeoutT, SyncTimeOut) for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Sync_req (i, PTCP_SYNC_PDU)	MASTER
11	SLAVE	**StartMaster_req ()** => Status := WRONG_SEQUENCE StartMaster_cnf (Status) BMA_Start_req ()	SLAVE

#	Current State	Event /Condition =>Action	Next State
12	SLAVE	**StopMaster_req ()** => Status := OK BMA_Stop_req () StopMaster_cnf (Status)	IDLE
13	SLAVE	**Sync_cnf ()** => ignore	SLAVE
14	SLAVE	**BMA_State_ind (State)** /State == SLAVE => ignore	SLAVE
15	SLAVE	**BMA_State_ind (State)** /State == MASTER => State := SINGLE_MASTER SequenceID++ StartTimer (SyncTimer, SyncInterval) StartTimer (TimeoutT, SyncTimeOut) StopSlave_req () for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Sync_req (i, PTCP_SYNC_PDU)	MASTER
16	MASTER	**StartMaster_req ()** => Status := WRONG_SEQUENCE StartMaster_cnf (Status)	MASTER
17	MASTER	**StopMaster_req ()** => Status := OK StopTimer (SyncTimer) StopTimer (TimeoutT) BMA_Stop_req () StopMaster_cnf (Status)	IDLE
18	MASTER	**BMA_State_ind (State)** /State == SLAVE => StopTimer (SyncTimer) StopTimer (TimeoutT) StartSlave_req ()	SLAVE
19	MASTER	**BMA_State_ind (State)** /State == MASTER => ignore	MASTER
20	MASTER	**Sync_ind ()** /!RECEIVE_PORT_VALID => ignore	MASTER
21	MASTER	**Sync_ind ()** /RECEIVE_PORT_VALID && !NEW_SYNC_FRAME => ignore	MASTER
22	MASTER	**Sync_ind ()** /RECEIVE_PORT_VALID && NEW_SYNC_FRAME => MultipleMaster := TRUE	MASTER
23	MASTER	**Sync_cnf ()** => ignore	MASTER

#	Current State	Event /Condition =>Action	Next State
24	MASTER	**TimerExpired (SyncTimer)** => SequenceID++ StartTimer (SyncTimer, SyncInterval) for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Sync_req (i, PTCP_SYNC_PDU)	MASTER
25	MASTER	**TimerExpired (TimeoutT)** /!MultipleMaster => StartTimer (TimeoutT, SyncTimeOut)	MASTER
26	MASTER	**TimerExpired (TimeoutT)** /MultipleMaster => MultipleMaster := FALSE StartTimer (TimeoutT, SyncTimeOut) SyncStateChange_ind (MULTIPLE_MASTER)	M_MASTER
27	M_MASTER	**StartMaster_req ()** => Status := WRONG_SEQUENCE StartMaster_cnf (Status)	M_MASTER
28	M_MASTER	**StopMaster_req ()** => Status := OK StopTimer (SyncTimer) StopTimer (TimeoutT) BMA_Stop_req () StopMaster_cnf (Status)	IDLE
29	M_MASTER	**BMA_State_ind (State)** /State == SLAVE => StopTimer (SyncTimer) StopTimer (TimeoutT) StartSlave_req ()	SLAVE
30	M_MASTER	**BMA_State_ind (State)** /State == MASTER => ignore	M_MASTER
31	M_MASTER	**Sync_ind ()** /!RECEIVE_PORT_VALID => ignore	M_MASTER
32	M_MASTER	**Sync_ind ()** /RECEIVE_PORT_VALID && !NEW_SYNC_FRAME => ignore	M_MASTER
33	M_MASTER	**Sync_ind ()** /RECEIVE_PORT_VALID && NEW_SYNC_FRAME => MultipleMaster := TRUE	M_MASTER
34	M_MASTER	**Sync_cnf ()** => ignore	M_MASTER

#	Current State	Event /Condition =>Action	Next State
35	M_MASTER	**TimerExpired (SyncTimer)** => SequenceID++ StartTimer (SyncTimer, SyncInterval) for i := all op. D_Ports do if (TRANSMIT_PORT_VALID) Sync_req (i, PTCP_SYNC_PDU)	M_MASTER
36	M_MASTER	**TimerExpired (TimeoutT)** /!MultipleMaster => StartTimer (TimeoutT, SyncTimeOut) SyncStateChange_ind (SINGLE_MASTER)	MASTER
37	M_MASTER	**TimerExpired (TimeoutT)** /MultipleMaster => MultipleMaster := FALSE StartTimer (TimeoutT, SyncTimeOut)	M_MASTER

4.4.4.5.5 Functions, Macros, Timers and Variables

Table 184 contains the functions, macros, timers and variables used by the SYN_MPSM and their arguments and their descriptions.

Table 184 – Functions, Macros, Timers and Variables used by the SYN_MPSM

Name	Type	Function/Meaning
StartTimer	Function	This local function starts or restarts a timer.
StopTimer	Function	This local function stops a timer.
TimerExpired	Function	This local function is issued if a timer expires.
NEW_SYNC_FRAME	Macro	Table 189 shall be applied.
RECEIVE_PORT_VALID	Macro	Table 194 shall be applied.
Sync_cnf (+/-) (D_Port)	Macro	LMPM_P_Data.cnf (CREP, D_Port, Tstamp, LMPM_status) Assignments: Status := LMPM_status
Sync_ind (S_Port, PTCP_SYNC_PDU)	Macro	Receive PTCP-PDU according to PTCP_SYNC_PDU. LMPM_P_Data.ind (CREP, S_Port, Tstamp, DA, SA, A_SDU) Assignments: PTCP_SYNC_PDU := A_SDU without LT and FrameID

Name	Type	Function/Meaning
Sync_req (D_Port, PTCP_SYNC_PDU)	Macro	Create PTCP-PDU according to PTCP_SYNC_PDU. Assignments: D_Port := D_Port DA := Multicast Address for PTCP_SYNC_PDU and corresponding SyncID SA := port source address PTCP_SYNC_PDU.SequenceID := SequenceID PTCP_SYNC_PDU.DomainUUID := DomainUUID PTCP_SYNC_PDU.MasterSourceAddress := local master address PTCP_SYNC_PDU.Time := local time from source PTCP_SYNC_PDU.TimeExtension := CurrentUTCOffset := Current UTC Offset PTCP_SYNC_PDU.MasterPriority1 := Priority PTCP_SYNC_PDU.ClockClass := Class PTCP_SYNC_PDU.ClockAccuracy := Accuracy PTCP_SYNC_PDU.ClockVariance := Variance PTCP_SYNC_PDU.MasterPriority2 := 0xFF A_SDU := LT, FrameID, PTCP_SYNC_PDU LMPM_P_Data.req (CREP, D_Port, Tstamp, DA, SA, A_SDU)
TRANSMIT_PORT_VALID	Macro	Table 195 shall be applied.

4.4.4.6 PTCP Slave Protocol Machine

4.4.4.6.1 Primitive definitions

4.4.4.6.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by PTCP Slave Protocol Machine (SYN_SPSM) are described in the service definition and shown in Table 185.

Table 185 – Remote primitives issued or received by SYN_SPSM

Primitive	Source	Destination	Associated parameters	Functions
LMPM_P_Data.ind	LMPM	SPSM	CREP, S_Port, Tstamp, DA, SA, A_SDU	—

4.4.4.6.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by SYN_SPSM are described in the service definition and shown in Table 186.

Table 186 – Local primitives issued or received by SYN_SPSM

Primitive	Source	Destination	Associated parameters	Functions
StartSlave_req	MPSM	SPSM	—	Activate slave function
StartSlave_cnf	SPSM	MPSM	Status	Confirmation of start slave
StopSlave_req	MPSM	SPSM	—	Deactivate slave function
StopSlave_cnf	SPSM	MPSM	Status	Confirmation of stop slave
SyncStateChange_ind	SPSM	FSPM	Status	Indication of events – JITTER_OUT_OF_BOUNDARY – NO_SYNC_MESSAGE_RECEIVED

Primitive	Source	Destination	Associated parameters	Functions
SLAVE_State_ind	SPSM	BMA	Status	Indication of events used by the best master algorithms – MASTER_LOST

4.4.4.6.2 State transition diagram

The state transition diagram of the SYN_SPSM is shown in Figure 61.

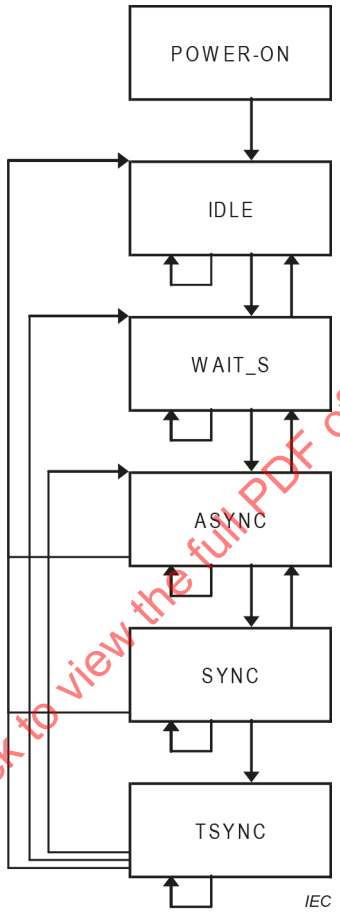


Figure 61 – State transition diagram of SYN_SPSM

States of the SYN_SPSM are:

POWER-ON	Data initialization
IDLE	The ASE service “StartSlave_req” shall start the slave function of the device.
WAIT_S	The slave is waiting for receiving sync messages in state WAIT_S.
ASYNC	The slave is not synchronized to the active master of the sync domain.
SYNC	The slave has achieved synchronicity with the active master of the sync domain.
TSYNC	The slave is still waiting for new sync messages.

4.4.4.6.3 State machine description

The slave protocol machine controls the transition of a sync slave device. Therefore an instance of the state machine exists for every SyncID with the key attribute PTCP_SubdomainUUID.

After startup the data management shall be initialized (POWER-ON) and the synchronization role shall be set (IDLE). The role is set by PTCP services. After this the SPSM changes from state IDLE to state WAIT_S. In state WAIT_S the slave waits for sync messages to synchronize to an active master. When receiving sync messages, the SPSM changes from state WAIT_S to state ASYNC. If the slave is synchronized the SPSM shall switch to state SYNC. If in state SYNC no sync message is received the SPSM shall switch to state TSYNC.

4.4.4.6.4 SYN_SPSM state table

Table 187 contains the state table used by SYN_SPSM.

Table 187 – SYN_SPSM state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> Stored PTCP_MasterSourceAddress := NIL	IDLE
2	IDLE	**StartSlave_req ()** => store Parameter Status := OK StartTimer (TakeoverTimeoutT, MasterTimeOut) StartSlave_cnf (Status)	WAIT_S
3	IDLE	**StopSlave_req ()** => Status := OK StopSlave_cnf (Status)	IDLE
4	WAIT_S	**StartSlave_req ()** => Status := WRONG_SEQUENCE StartSlave_cnf (Status)	WAIT_S
5	WAIT_S	**StopSlave_req ()** => Status := OK StopTimer (TakeoverTimeoutT) StopSlave_cnf (Status)	IDLE
6	WAIT_S	**Sync_ind ()** /!VALID_SYNC_FRAME => ignore	WAIT_S
7	WAIT_S	**Sync_ind ()** /VALID_SYNC_FRAME => L_Time := LocalTime M_Time := PTCP_SYNC_DATA.Time OFFSET_CTRL_START (L_Time, M_Time) StartTimer (OffsetContLoopT, SyncInterval) StartTimer (TakeoverTimeoutT, MasterTimeOut) StartTimer (SyncTimeoutT, SyncTimeOut) SyncStateChange_ind (JITTER_OUT_OF_BOUNDARY)	ASYNC
8	WAIT_S	**TimerExpired (TakeoverTimeoutT)** => Stored PTCP_MasterSourceAddress := NIL SLAVE_State_ind (MASTER_LOST)	WAIT_S
9	ASYNC	**StartSlave_req ()** => Status := WRONG_SEQUENCE StartSlave_cnf (Status)	ASYNC

#	Current State	Event /Condition =>Action	Next State
10	ASYNC	**StopSlave_req ()** => OFFSET_CTRL_STOP Status := OK StopTimer (OffsetContLoopT) StopTimer (TakeoverTimeoutT) StopTimer (SyncTimeoutT) StopSlave_cnf (Status)	IDLE
11	ASYNC	**Sync_ind ()** /!VALID_SYNC_FRAME => ignore	ASYNC
12	ASYNC	**Sync_ind ()** /VALID_SYNC_FRAME => L_Time := LocalTime M_Time := PTCP_SYNC_DATA.Time StartTimer (TakeoverTimeoutT, MasterTimeOut) StartTimer (SyncTimeoutT, SyncTimeOut) SyncStateChange_ind (INFO, M_Time, L_Time)	ASYNC
13	ASYNC	**TimerExpired (OffsetContLoopT)** /!OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (L_Time, M_Time) StartTimer (OffsetContLoopT, SyncInterval)	ASYNC
14	ASYNC	**TimerExpired (OffsetContLoopT)** /OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (L_Time, M_Time) StartTimer (OffsetContLoopT, SyncInterval) SyncStateChange_ind (SYNC)	SYNC
15	ASYNC	**TimerExpired (TakeoverTimeoutT)** => Stored PTCP_MasterSourceAddress := NIL StopTimer (SyncTimeoutT) SLAVE_State_ind (MASTER_LOST)	WAIT_S
16	SYNC	**StartSlave_req ()** => Status := WRONG_SEQUENCE StartSlave_cnf (Status)	SYNC
17	SYNC	**StopSlave_req ()** => OFFSET_CTRL_STOP Status := OK StopTimer (OffsetContLoopT) StopTimer (TakeoverTimeoutT) StopTimer (SyncTimeoutT) StopSlave_cnf (Status)	IDLE
18	SYNC	**Sync_ind ()** /!VALID_SYNC_FRAME => ignore	SYNC

#	Current State	Event /Condition =>Action	Next State
19	SYNC	**Sync_ind ()** /VALID_SYNC_FRAME => L_Time := LocalTime M_Time := PTCP_SYNC_DATA.Time StartTimer (TakeoverTimeoutT, MasterTimeOut) StartTimer (SyncTimeoutT, SyncTimeOut) SyncStateChange_ind (INFO, M_Time, L_Time)	SYNC
20	SYNC	**TimerExpired (OffsetContLoopT)** /OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (L_Time, M_Time) StartTimer (OffsetContLoopT, SyncInterval)	SYNC
21	SYNC	**TimerExpired (OffsetContLoopT)** /!OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (L_Time, M_Time) StartTimer (OffsetContLoopT, SyncInterval) SyncStateChange_ind (JITTER_OUT_OF_BOUNDARY) // NOTE Signal to SYNC_RELAY "Stored PTCP_MasterSourceAddress := NIL"	ASYNC
22	SYNC	**TimerExpired (TakeoverTimeoutT)** => Stored PTCP_MasterSourceAddress := NIL SLAVE_State_ind (MASTER_LOST)	TSYNC
23	TSYNC	**StartSlave_req ()** => Status := WRONG_SEQUENCE StartSlave_cnf (Status)	TSYNC
24	TSYNC	**StopSlave_req ()** => OFFSET_CTRL_STOP Status := OK StopTimer (OffsetContLoopT) StopTimer (TakeoverTimeoutT) StopTimer (SyncTimeoutT) StopSlave_cnf (Status)	IDLE
25	TSYNC	**Sync_ind ()** /!VALID_SYNC_FRAME => ignore	TSYNC
26	TSYNC	**Sync_ind ()** /VALID_SYNC_FRAME => L_Time := LocalTime M_Time := PTCP_SYNC_DATA.Time StartTimer (TakeoverTimeoutT, MasterTimeOut) StartTimer (SyncTimeoutT, SyncTimeOut) SyncStateChange_ind (INFO, M_Time, L_Time)	ASYNC
27	TSYNC	**TimerExpired (OffsetContLoopT)** => StartTimer (OffsetContLoopT, SyncInterval) //NOTE The control loop should try to keep the clock stable	TSYNC
28	TSYNC	**TimerExpired (SyncTimeoutT)** => OFFSET_CTRL_STOP StopTimer (OffsetContLoopT) SyncStateChange_ind (NO_SYNC_MESSAGE_RECEIVED)	WAIT_S

4.4.4.6.5 Functions, Macros, Timers and Variables

Table 188 contains the functions, macros, timers and variables used by the SYN_SPSM and their arguments and their descriptions.

Table 188 – Functions, Macros, Timers and Variables used by the SYN_SPSM

Name	Type	Function/Meaning
StartTimer	Function	This local function starts or restarts a timer.
StopTimer	Function	This local function stops a timer.
TimerExpired	Function	This local function is issued if a timer expires.
OFFSET_CTRL_CALC	Macro	This local macro calculates the offset compensation Parameters: – LocalTime – MasterTime
OFFSET_CTRL_IS_SYNC	Macro	This local macro measures the sync error in PLL Window. It shall only return TRUE if remaining SYNC has a high probability. //NOTE The detection of a fault depends on the filter algorithm of the selected control loop, because the decision is made with local information only.
OFFSET_CTRL_START	Macro	This local macro initializes the control loop for offset compensation Parameters: – LocalTime – MasterTime
OFFSET_CTRL_STOP	Macro	This local macro stops the offset compensation.
Sync_ind	Macro	Receive PTCP-PDU according to PTCP_SYNC_PDU. LMPM_P_Data.ind (CREP, S_Port, Tstamp, DA, SA, A_SDU) if A_SDU.FrameID == WITHOUT_FUP_PDU PTCP_SYNC_DATA := PTCP_SYNC_PDU else wait for valid PTCP_FUP_PDU from active Master LMPM_P_Data.ind (CREP, S_Port, DA, SA, A_SDU) PTCP_SYNC_DATA.Delay += PTCP_FUP_PDU.Delay Assignments: PTCP_SYNC_PDU := A_SDU without LT and FrameID PTCP_FUP_PDU := A_SDU without LT and FrameID Parameters: – SourcePort – LocalTime – PTCP_SYNC_DATA
VALID_SYNC_FRAME	Macro	Table 189 shall be applied. Additionally a valid RATE shall exist.
OffsetContLoopT	Timer	This local timer is used for the offset checking between LocalTime and MasterTime. It shall be aligned to the SyncInterval for best control loop results.
SyncTimeoutT	Timer	This local timer is used to monitor the existence of the sync master. If this time expires the sync master is lost.
TakeoverTimeoutT	Timer	This local timer is used to monitor the existence of the sync master. If this time expires the preparation for a master switchover is done. This time shall always be less than SyncTimeoutT.
MasterTimeout	Variable	This local variable contains the value for the warning / preparation level master timeout. Default value: 3 x SyncInterval //NOTE This value is derived from the PTCPTakeoverTimeoutFactor

Name	Type	Function/Meaning
PTCP_MasterSourceAddress	Variable	This shared variable contains the MasterSourceAddress of the actual SyncMaster. It is additionally changed by the SYNC_RELAY. The MasterSourceAddress is stored whenever the SYN_SPSM enters the SYNC state.
Status	Variable	This local variable is used to store the status for further use.
SyncInterval	Variable	This local variable contains the value of the sync indication receiving interval. Default value: SyncInterval //NOTE This value is derived from the SyncSendFactor.
SyncTimeout	Variable	This local variable contains the value fault level master timeout. Default value: 6 x SyncInterval //NOTE This value is derived from the PTCPTimeoutFactor.

4.4.4.6.6 Sync receiving rules truth table for one SyncID

Table 189 contains the truth table used by the SPSM.

Table 189 – Truth table for one SyncID for receiving sync and follow up frames

Input								Output
Condition of the receive port [b]								Local
Port State [d]	MAU-Type [f]	Line Delay	Ingress Filter	Sub-domain UUID	Master MAC-Address	PTCP_-Sequence ID	PTCP_Delay1ns / PTCP_Delay10ns	Frame from port valid
Disabled	—	—	—	—	—	—	—	No
!Disabled	Wrong	—	—	—	—	—	—	No
!Disabled	Ok	No	—	—	—	—	—	No
!Disabled	Ok	Yes	Yes	—	—	—	—	No
!Disabled	Ok	Yes	No	Wrong	—	—	—	No
!Disabled	Ok	Yes	No	Equal [a]	Wrong	—	—	No
!Disabled	Ok	Yes	No	Equal [a]	Equal [a]	Older	—	No
!Disabled	Ok	Yes	No	Equal [a]	Equal [a]	Newer [c]	Invalid [e]	No
!Disabled	Ok	Yes	No	Equal [a]	Equal [a]	Newer [c]	Valid [e]	Yes

[a] If the own PTCP_SubdomainUUID and/or the MasterMACAddress is unknown (coded as zero), the compare with the values of the PTCP sync frame may return “equal” to optimize the startup of the synchronization.

[b] At the receive port, the local PTCP_SubdomainUUID and/or the MasterMACAddress is checked against the PTCP frame.

[c] If the own MasterMACAddress is unknown (coded as zero), a PTCP sync frame with the local PTCP_SubdomainUUID is defined as ‘newer’. The PTCP_SequenceID is bound to the MasterMACAddress.

[d] According to IEEE Std 802.3 the PortStates are Disabled, Discarding, Learning and Forwarding. Only the PortState Disabled allows no forwarding or receiving of frames.

[e] See the definition of the fields PTCP_Delay10ns and PTCP_Delay1ns for details.

[f] MAUType supports full duplex and a link exists.

4.4.4.7 Sync Relay Protocol Machine

4.4.4.7.1 Primitive definitions

4.4.4.7.1.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Sync Relay Protocol Machine (SYNC_RELAY) are described in the service definition and shown in Table 190.

Table 190 – Remote primitives issued or received by SYNC_RELAY

Primitive	Source	Destination	Associated parameters	Functions
LMPM_P_Data.req ()	SYNC_RELAY	LMPM	CREP, D_Port, Tstamp, DA, SA, A_SDU	—
LMPM_P_Data.cnf ()	LMPM	SYNC_RELAY	CREP, D_Port, Tstamp, LMPM_status	—
LMPM_P_Data.ind ()	LMPM	SYNC_RELAY	CREP, D_Port, Tstamp, DA, SA, A_SDU	—

4.4.4.7.1.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by SYNC_RELAY are described in the service definition and shown in Table 191.

Table 191 – Local primitives issued or received by SYNC_RELAY

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.4.4.7.2 State transition diagram

The state transition diagram of the SYNC_RELAY is shown in Figure 62.

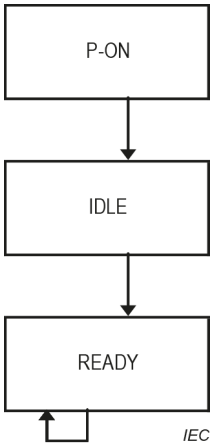


Figure 62 – State transition diagram of SYNC_RELAY

States of the SYNC_RELAY are:

P-ON	Initialization of the Filtering Data Base for sync frames.
IDLE	Start of the receipt timer for aging the entries in the Filtering Data Base.
READY	The state READY describes the rules for forwarding sync and follow up messages.

4.4.4.7.3 State machine description

The SYNC_RELAY protocol machine describes the sync and follow up forwarding rules for bridges. It is almost independent from the role PTCP master or PTCP slave and thus applies to both.

NOTE A SYNC_RELAY of a PTCP master never forwards sync messages created by the local PTCP master.

Each timestamp unit of the bridge shall instantly correct the delay field of sync frames. The time correction (bridge and line delay) is done by adding the delay while transmitting a sync frame.

The SYNC_RELAY protocol machine is necessary to avoid sync messages circulating in the network.

Announce frames shall be handled by the MAC_RELAY. In addition to the MAC_RELAY rules an implicit egress and ingress boundary should be activated for every port which does not support line delay measurement.

For a high precision the bridge delay of the sync and the follow up frame should be small and stable.

4.4.4.7.4 SYNC_RELAY state table

Table 192 contains the state table used by SYNC_RELAY. Only the PTCP frames from one master and one PTCP_SubdomainUUID for one SyncID shall be forwarded. After the selection of a master, the PTCP frames of the others shall be dropped. Table 194 and Table 195 show the conditions at a glance.

Table 192 – SYNC_RELAY state table

#	Current State	Event /Condition =>Action	Next State
1	P-ON	=> Init RelayTable	IDLE
2	IDLE	=> StartTimer (PTCP_Timeout) StartTimer (PTCP_TakeoverTimeout)	READY
3	READY	**TimerExpired (PTCP_Timeout)** => AGING_OF_FWSM_LIST(PT) StartTimer (PTCP_Timeout)	READY
4	READY	**TimerExpired (PTCP_TakeoverTimeout)** => AGING_OF_FWSM_LIST(PTT) StartTimer (PTCP_TakeoverTimeout)	READY
5	READY	**Sync_Ind (S_Port, PTCP_SYNC_PDU)** /!RECEIVE_PORT_VALID => ignore	READY
6	READY	**Sync_Ind (S_Port, PTCP_SYNC_PDU)** /RECEIVE_PORT_VALID && !NEW_SYNC_FRAME => ignore	READY
7	READY	**Sync_Ind (S_Port, PTCP_SYNC_PDU)** /RECEIVE_PORT_VALID && NEW_SYNC_FRAME && !RATE_VALID => UPDATE_MASTER_ENTRY CalculateMasterRCF	READY

8	READY	**Sync_Ind (S_Port, PTCP_SYNC_PDU)**
		/RECEIVE_PORT_VALID
		&& NEW_SYNC_FRAME
		&& RATE_VALID
		=>
		UPDATE_MASTER_ENTRY
		CalculateMasterRCF
		for i := all op. D_Ports \ S_Port
		do
		if (TRANSMIT_PORT_VALID)
		Sync_Req (i, PTCP_SYNC_PDU)
		
		//NOTE If the hardware does not support “sync request only forwarding” then a Sync-Frame and a FollowUp-Frame are conveyed.
			READY

9	READY	**Sync_Cnf ()** => ignore	READY
10	READY	**Followup_Ind (S_Port, PTCP_FUP_PDU)** /!RECEIVE_PORT_VALID => ignore	READY
11	READY	**Followup_Ind (S_Port, PTCP_FUP_PDU)** /RECEIVE_PORT_VALID && !VALID_FU_FRAME => ignore	READY

#	Current State	Event /Condition =>Action	Next State
12	READY	**Followup_Ind (S_Port, PTCP_FUP_PDU)** /RECEIVE_PORT_VALID && VALID_FU_FRAME && RATE_VALID => SET_FU_RECEIVED for i := all op. D_Ports \ S_Port do if (TRANSMIT_PORT_VALID) Followup_Req (i, PTCP_FUP_PDU)	READY
13	READY	**Followup_Cnf ()** => ignore	READY

4.4.4.7.5 Functions, Macros, Timers and Variables

Table 193 contains the functions, macros, timers and variables used by the SYNC_RELAY and their arguments and their descriptions.

Table 193 – Functions, Macros, Timers and Variables used by the SYNC_RELAY

Name	Type	Function/Meaning
CalculateMasterRCF	Function	This local function calculates and updates the rate ($RCF_{SyncMaster}$)
AGING_OF_FWSM_LIST	Macro	This local macro shall handle the entry in the RelayTable. Only none or one entry exists. PTCP_Timeout expired (PT): Sync master lost. Thus the stored sync master, PTCP_SubdomainUUID (if not defined locally) and rate is deleted. PTCP_TakeoverTimeout expired (PTT): Wait for primary/secondary sync master switchover. Thus the stored MasterMACAddress is deleted. RelayTableEntry[SyncID, PTCP_SubdomainUUID]: – MasterAddress – SequenceID – RxPort – FuFrame Case “PTCP_SubdomainUUID is NOT parameterized”: An entry is made automatically for the first PTCP_Sync frame received via a valid link. The entry is discarded if a different PTCP_SubdomainUUID is parameterized later. Case “PTCP_SubdomainUUID is parameterized” An entry is made automatically for the first PTCP_Sync frame received via a valid link with the correct PTCP_SubdomainUUID.
Followup_Cnf	Macro	LMPM_P_Data.cnf (CREP, D_Port, Tstamp, LMPM_status) Assignments: Status := LMPM_status Parameters: – DestinationPort – Status

Name	Type	Function/Meaning
Followup_Ind	Macro	Receive PTCP-PDU according to PTCP_FUP_PDU LMPM_P_Data.ind (CREP, S_Port, Tstamp, DA, SA, A_SDU) Assignments: PTCP_FUP_PDU := A_SDU without LT and FrameID Parameters: – SourcePort – PTCP_FUP_PDU
Followup_Req	Macro	Create PTCP-PDU according to PTCP_FUP_PDU Assignments: D_Port := Port DA := multicast address for PTCP_FUP_PDU SA := local port address A_SDU := LT, FrameID, PTCP_FUP_PDU LMPM_P_Data.req (CREP, D_Port, Tstamp, DA, SA, A_SDU) Parameters: – DestinationPort – PTCP_FUP_PDU
NEW_SYNC_FRAME	Macro	Table 189 shall be applied.
RATE_VALID	Macro	This local macro checks if the rate ($RCF_{SyncMaster}$) is calculated and in the range of $\pm 250 \frac{\mu Hz}{Hz}$.
RECEIVE_PORT_VALID	Macro	Table 194 shall be applied.
SET_FU_RECEIVED	Macro	Update entry in RelayTable[SyncID, DomainUUID] FuFrame := TRUE
Sync_Cnf	Macro	LMPM_P_Data.cnf (CREP, D_Port, Tstamp, LMPM_status) Assignments: Status := LMPM_status Parameters: – DestinationPort – Status
Sync_Ind	Macro	Receive PTCP-PDU according to PTCP_SYNC_PDU LMPM_P_Data.ind (CREP, S_Port, Tstamp, DA, SA, A_SDU) Assignments: PTCP_SYNC_PDU := A_SDU without LT and FrameID Parameters: – SourcePort – PTCP_SYNC_PDU
Sync_Req	Macro	Create PTCP-PDU according to PTCP_SYNC_PDU Assignments: D_Port := Port DA := multicast address for PTCP_SYNC_PDU SA := local port address A_SDU := LT, FrameID, PTCP_SYNC_PDU LMPM_P_Data.req (CREP, D_Port, Tstamp, DA, SA, A_SDU) Parameters: – DestinationPort – PTCP_SYNC_PDU
TRANSMIT_PORT_VALID	Macro	Table 195 shall be applied.

Name	Type	Function/Meaning
UPDATE_MASTER_ENTRY	Macro	Update entry in RelayTable[SyncID, DomainUUID] for sync master MasterAddress := PTCP_SYNC_PDU.MasterSourceAddress SequenceID := PTCP_SYNC_PDU.SequenceID RxPort := S_Port FuFrame := FALSE
VALID_FU_FRAME	Macro	Check RelayTable[SyncID, DomainUUID] for valid receive port and sequence number MasterAddress == PTCP_FUP_PDU.MasterSourceAddress SequenceID == PTCP_FUP_PDU.SequenceID RxPort == S_Port FuFrame == FALSE
PTCP_TakeoverTimeout	Timer	This local variable contains the value for the warning / preparation level master timeout. Default value: 2 x SyncInterval (Sync Slave) 3 x SyncInterval (Sync Master) //NOTE This value is derived from the PTCPTakeoverTimeoutFactor.
PTCP_Timeout	Timer	This local variable contains the value fault level master timeout. Default value: 6 x SyncInterval //NOTE This value is derived from the PTCPTimeoutFactor.

4.4.4.7.6 Sync forwarding rules for bridges truth table for one SyncID

Table 194 and Table 195 contain the truth table used by the SYNC_RELAY.

Table 194 – Truth table for one SyncID for receiving

Input						Output
Condition of the receive port [b]						Local
PortState [c]	MAUType	LineDelay	Ingress-Filter	Subdomain-UUID	MasterMACAddress	Frame from port valid
Disabled	—	—	—	—	—	No
!Disabled	Wrong	—	—	—	—	No
!Disabled	Ok	No	—	—	—	No
!Disabled	Ok	Yes	Yes	—	—	No
!Disabled	Ok	Yes	No	Wrong	—	No
!Disabled	Ok	Yes	No	Equal [a]	Wrong	No
!Disabled	Ok	Yes	No	Equal [a]	Equal [a]	Yes [d]

[a] If the own PTCP_SubdomainUUID and/or the MasterMACAddress is unknown (coded as zero), the compare with the values of the PTCP sync frame may return “equal” to optimize the startup of the synchronization.

[b] At the receive port, the local PTCP_SubdomainUUID and/or the MasterMACAddress is checked against the PTCP frame.

[c] According to IEEE Std 802.3 the PortStates are Disabled, Discarding, Learning and Forwarding. Only the PortState Disabled allows no forwarding or receiving of frames.

[d] An optional filter should be used to discard frames with invalid content from the PTCP_Delay1ns / PTCP_Delay10ns fields.

Table 195 – Truth table for one SyncID for transmitting

Input							Output
Local	Conditions of a transmit port [b]						Local
Rate known	Subdomain-UUID	MasterMAC-Address	PortState	MAU-Type	Line Delay	Egress-Filter	Port for frame valid
No	—	—	—	—	—	—	No
Yes	Wrong	—	—	—	—	—	No
Yes	Equal [a]	Wrong	—	—	—	—	No
Yes	Equal [a]	Equal [a]	Discarding	—	—	—	No
Yes	Equal [a]	Equal [a]	! Discarding	Wrong	—	—	No
Yes	Equal [a]	Equal [a]	! Discarding	Ok	No	—	No
Yes	Equal [a]	Equal [a]	! Discarding	Ok	Yes	Yes	No
Yes	Equal [a]	Equal [a]	! Discarding	Ok	Yes	No	Yes

[a] If the own PTCP_SubdomainUUID and/or the MasterMACAddress is unknown (coded as zero), the compare with the values of the PTCP frame may return “equal” to optimize the startup of the synchronization.

[b] At the transmit port, the remote PTCP_SubdomainUUID and/or the MasterMACAddress is checked against the PTCP frame.

4.4.4.8 Scheduler Protocol Machine

4.4.4.8.1 Primitive definitions

4.4.4.8.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by SCHEDULER are described in the service definition and shown in Table 196.

Table 196 – Remote primitives issued or received by SCHEDULER

Primitive	Source	Destination	Associated parameters	Functions
LMPM_C_Data.req	SCHEDULER	LMPM	CREP, PortList, DA, SA, Prio, C_SDU, APDU_Status	—
UDP_C_Data.req	SCHEDULER	IP	IPPort, CREP, DA, SA, Prio, C_SDU, APDU_Status	—
LMPM_C_Data.cnf	LMPM	SCHEDULER	CREP, PortList, DA, SA, Prio, C_SDU, APDU_Status	—
UDP_C_Data.cnf	IP	SCHEDULER	IPPort, CREP, DA, SA, Prio, C_SDU, APDU_Status	—

4.4.4.8.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by SCHEDULER are described in the service definition and shown in Table 197.

Table 197 – Local primitives issued or received by SCHEDULER

Primitive	Source	Destination	Associated parameters	Functions
DMUX_Schedule_cnf	DEMUX	SCHEDULER	—	—
MUX_Schedule_cnf	MUX	SCHEDULER	—	—
DMUX_Schedule_req	SCHEDULER	DEMUX	CycleCounter, Rx_Period	—
DMUX_Schedule_req	SCHEDULER	DEMUX	Rx_Period	—
MUX_Schedule_req	SCHEDULER	MUX	CycleCounter, Tx_Period	—
MUX_Schedule_req	SCHEDULER	MUX	Tx_Period	—
SCHEDULER_add_req	PPM	SCHEDULER	CREP, ReductionRatio, Phase, Sequence, TxOptions	Add a PPM to the scheduler
SCHEDULER_remove_req	PPM	SCHEDULER	CREP	Remove a PPM from the scheduler
SCHEDULER_error_ind	SCHEDULER	CMSU CTLSU	CREP, ErrorCode	Signal an overload. In this case the NEXT phase shall be skipped
SCHEDULER_add_cnf	SCHEDULER	PPM	CREP	Add a PPM to the scheduler
SCHEDULER_remove_cnf	SCHEDULER	PPM	CREP	Remove a PPM from the scheduler

4.4.4.8.2 State transition diagram

The state transition diagram of the SCHEDULER is shown in Figure 63.

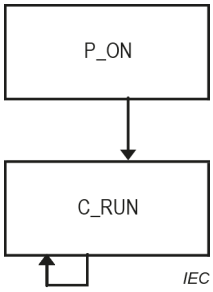


Figure 63 – State transition diagram of SCHEDULER

States of the SCHEDULER are:

P_ON	Data initialization
C_RUN	Scheduler is active and running

4.4.4.8.3 State machine description

The SCHEDULER state machine generates, based on the WorkingClock, TickEvent signals for the local scheduling of frames according to the SendListControl. Each PPM is registered at the SCHEDULER to provide the needed parameters. All RT_CLASS_x frames are handled by this state machine according to the sorting of the SendList.

Additionally, the RED_RELAY uses this TickEvent to trigger the timely forwarding of RT_CLASS_3 frames.

4.4.4.8.4 SCHEDULER state table

Table 198 contains the state table used by SCHEDULER.

Table 198 – SCHEDULER state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	**Scheduler_Init_req ()** => TxPeriod := GREEN RxPeriod := GREEN MUX_Schedule_req(TxPeriod) DMUX_Schedule_req(RxPeriod)	C_RUN
2	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /TxPeriod == GREEN && TxPeriodChecker () == GREEN => ignore	C_RUN
3	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /TxPeriod == RED && TxPeriodChecker () == GREEN => TxPeriod := GREEN MUX_Schedule_req(CycleCounter, TxPeriod)	C_RUN
4	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /TxPeriod == RED && TxPeriodChecker () == RED => ignore	C_RUN
5	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /TxPeriod == GREEN && TxPeriodChecker () == RED => TxPeriod := RED MUX_Schedule_req(CycleCounter, TxPeriod)	C_RUN
6	C_RUN	**MUX_Schedule_cnf()** => ignore	C_RUN
7	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /RxPeriod == GREEN && RxPeriodChecker () == GREEN => ignore	C_RUN
8	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /RxPeriod == RED && RxPeriodChecker () == GREEN => RxPeriod := GREEN DMUX_Schedule_req(CycleCounter, RxPeriod)	C_RUN
9	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /RxPeriod == RED && RxPeriodChecker () == RED => ignore	C_RUN

#	Current State	Event /Condition =>Action	Next State
10	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /RxPeriod == GREEN && RxPeriodChecker () == RED => RxPeriod := RED DMUX_Schedule_req(CycleCounter, RxPeriod)	C_RUN
11	C_RUN	**DMUX_Schedule_cnf()** => ignore	C_RUN
12	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /for each active PPM check whether CycleCounter, CycleOffset fits to FrameSendOffset and ReductionRatio check if PPM ready to schedule and frame not already at the LMPM => C_Data_req()	C_RUN
13	C_RUN	**TickEvent (CycleCounter, CycleOffset)** /for each active PPM check whether CycleCounter, CycleOffset fits to FrameSendOffset and ReductionRatio check if PPM ready to schedule and frame already at the LMPM => SCHEDULER_error.ind (ErrorCode = Overload)	C_RUN
14	C_RUN	**C_Data_cnf()** => ignore	C_RUN
15	C_RUN	**SCHEDULER_add.req (CREP, ReductionRatio, Phase, Sequence)** => Add PPM to list SCHEDULER_add.cnf (CREP)	C_RUN
16	C_RUN	**SCHEDULER_remove.req (CREP)** => Remove PPM from list SCHEDULER_remove.cnf (CREP)	C_RUN

4.4.4.8.5 Functions, Macros, Timers and Variables

Table 199 contains the functions, macros, timers and variables used by the SCHEDULER and their arguments and their descriptions.

Table 199 – Functions, Macros, Timers and Variables used by the SCHEDULER

Name	Type	Function/Meaning
TickEvent (CycleCounter, CycleOffset)	Function	Locally generated event from a timer. The timer shall be controlled by the PTCP master or slave in case of synchronization or free running without synchronization. See the definition of the field CycleCounter.
C_Data_cnf	Macro	if (txOption == RTC) LMPM_C_Data.cnf (CREP, LMPM_status) if (txOption == UDP) UDP_C_Data.cnf (IPPort, CREP, LMPM_status)
C_Data_req	Macro	if (txOption == RTC) LMPM_C_Data.req (CREP, Port, DA, SA, Prio, C_SDU, APDU_Status) if (txOption == UDP) UDP_C_Data.req (IPPort, CREP, DA, SA, Prio, C_SDU, APDU_Status)
TxPeriodChecker	Function	This local function checks the active TxPeriod and returns it.
RxPeriodChecker	Function	This local function checks the active RxPeriod and returns it.
RxPeriod	Variable	This variable contains the values: RED and GREEN

Name	Type	Function/Meaning
TxPeriod	Variable	This variable contains the values: RED and GREEN

4.4.4.8.6 Truth tables

4.4.4.8.6.1 General

Table 200 and Table 201 contain the truth tables used by the SCHEDULER to detect the required period per port.

4.4.4.8.6.2 Receive port PeriodChecker

Table 200 shows the truth table for a receive port.

Table 200 – Truth table for RxPeriodChecker of one port

Input		Output
RTC3PSM port state	CycleOffset inside RED period	RxPeriod
OFF	—	GREEN
—	Outside	GREEN
UP	Inside	GREEN
RUN	Inside	RED

4.4.4.8.6.3 Transmit port PeriodChecker

Table 201 shows the truth table for a transmit port.

Table 201 – Truth table for TxPeriodChecker of one port

Input		Output
RTC3PSM port state	CycleOffset inside RED period	TxPeriod
OFF	—	GREEN
—	Outside	GREEN
UP	Inside	RED
RUN	Inside	RED

4.4.5 DLL Mapping Protocol Machines

There is no DLL Mapping Protocol Machine (DMPM) defined for this protocol.

4.5 Time synchronization

4.5.1 General

Time synchronization is used to align the timescales between clocks (see Table 202) implemented in a master and its slaves. Two timescales are used in this document as shown in Table 203.

Table 202 – Alignment of terms to IEEE Std 802.1AS

Clocks	IEEE Std 802.1AS	Meaning
Local clock	LocalClock	Free running clock, which is not rate or offset compensated (not disciplined).
Application clock e.g., for Global Time or Working Clock	ClockTarget (slave) or ClockSource (master)	Disciplined clock, which is used by the application as source of time. It may not be disciplined in case of ClockSource.
PTP clock e.g., for Global Time or Working Clock	ClockSlave (slave) or ClockMaster (master)	Disciplined clock, which is used by the IEEE Std 802.1AS as source of time. It is disciplined by the ClockSource, even in case of ClockMaster.

Table 203 – Timescales

Timescale	Used protocol	Domain-Number	SyncID	Meaning
GlobalTime	IEEE Std 802.1 AS	0x00 and 0x01 (Redundant)	—	Used to provide the global understanding of Wall Clock time for the usage in this document.
Working-Clock	IEEE Std 802.1 AS	0x20 and 0x21 (Redundant)	—	Time-aware system Used in the context of time-aware stations. An arbitrary timescale used in a given domain to align a group of devices to a common working clock.
	PTCP	—	0x00	Non-time-aware system An arbitrary timescale used in a given domain to align a group of devices to a common working clock.

Figure 64 shows the integration model for synchronization based on different clocks. It shows just one timescale. The coupling control reacts to clock state changes of the ClockSlave which are acknowledged by the application.

The clocks have different purposes:

- LocalClock
  - Time strictly monotonic increasing
  - Used to measure line delay / cable delay / Pdelay
  - Used to measure neighbor rate ratio
- ClockSlave (or ClockMaster)
  - Time monotonic increasing
  - Working Clock: Rate can change up to 250 ns over an interval of 1 ms
  - Global Time: Rate can change up to 1 µs over an interval of 1 ms
  - Rate and offset compensated to follow a ClockMaster as good as possible
- ClockTarget (or ClockSource)
  - If coupled, follows the ClockSource as good as possible
  - If uncoupled, time monotonic increasing
  - If re-coupled by application, jumps to the actual value of the ClockSlave

– Working Clock: Error budget < 1 µs
– Global Time: Error budget < 1 ms

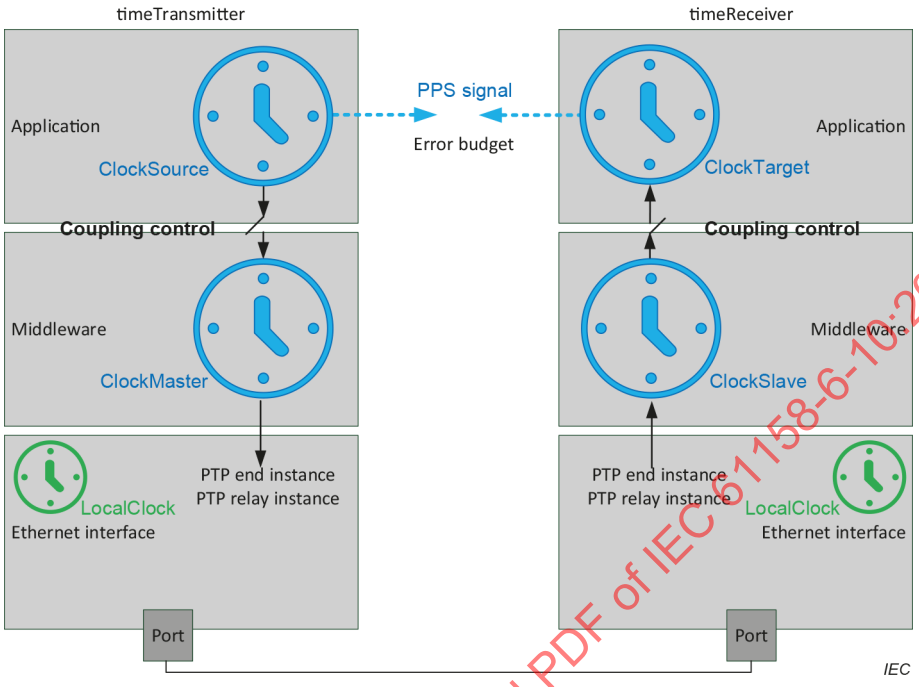


Figure 64 – Station clock model

Figure 65 shows the integration of time synchronization into an end station. Working Clock and Global Time shall be available for the application independent from the network.

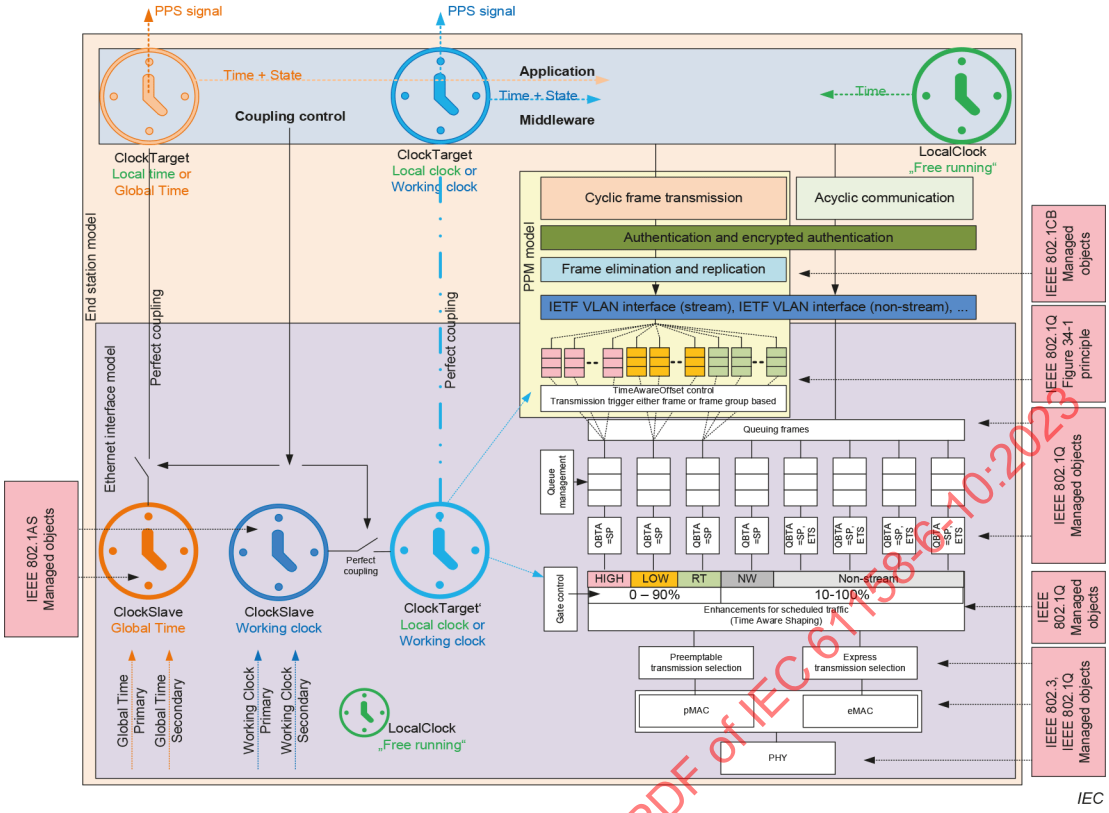


Figure 65 – End station model with time synchronization

4.5.2 GlobalTime

For time synchronization, for example synchronization of the local maintained GlobalTime, IEEE Std 802.1AS applies. Table 203 and Table 204 show the relation between the timescales and Figure 66 shows the used timer model for GlobalTime.

Table 204 – Timescale correspondence between GlobalTime, TAI and UTC

GlobalTime		TAI (International Atomic Time)	UTC (Coordinated Universal Time)
Value Seconds [48 bit]	Value Nanoseconds [32 bit]	Value	Value
0	0	1970-01-01 T 00:00:00	1969-12-31 T 23:59:51.999918 Z

NOTE The coding YYYY-MM-DD T HH:MM:SS Z is according to ISO 8601-1, where 'T' indicates that a time follows the date and 'Z' indicates that the time is UTC.

An 80 bit wide synchronized timer with a resolution of 1 ns is used for the GlobalTime independently from the assigned role master or slave. GlobalTime is the source or sink of time for role master or slave. It is the time source for timestamps. Together with the GlobalTime, a status (see Table 14) containing information about the quality of the GlobalTime is maintained.

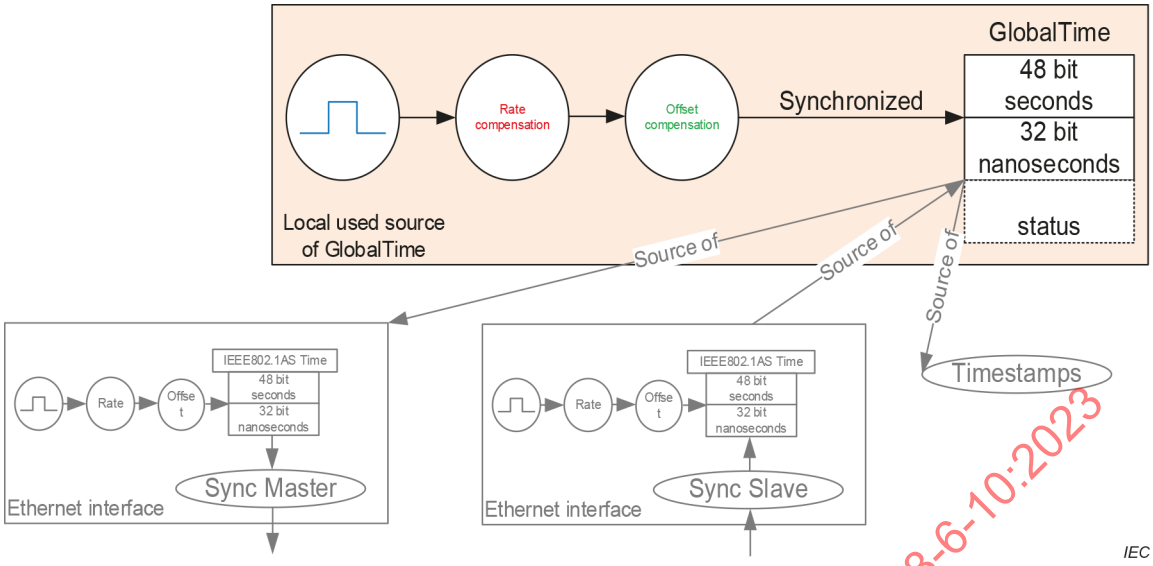


Figure 66 – GlobalTime timer model

4.5.3 WorkingClock

4.5.3.1 General

For working clock synchronization, for example synchronization of the local maintained WorkingClock, PTCP and IEEE Std 802.1AS apply. Table 203 and Table 205 show the used timescale and Figure 67 shows the used timer model for WorkingClock.

Table 205 – Timescale correspondence between WorkingClock, TAI and UTC

WorkingClock		TAI (International Atomic Time) Value	UTC (Coordinated Universal Time) Value
Value Seconds [48 bit]	Value Nanoseconds [32 bit]		
0	0	—	—

An 80 bit wide synchronized timer with a resolution of 1 ns is used for the WorkingClock independently from the assigned role master or slave. WorkingClock is the source or sink of working clock for role master or slave. It is the source for timestamps and signals which are bound to the WorkingClock. Together with the WorkingClock, a status (see Table 14) containing information about the quality of the WorkingClock is maintained.

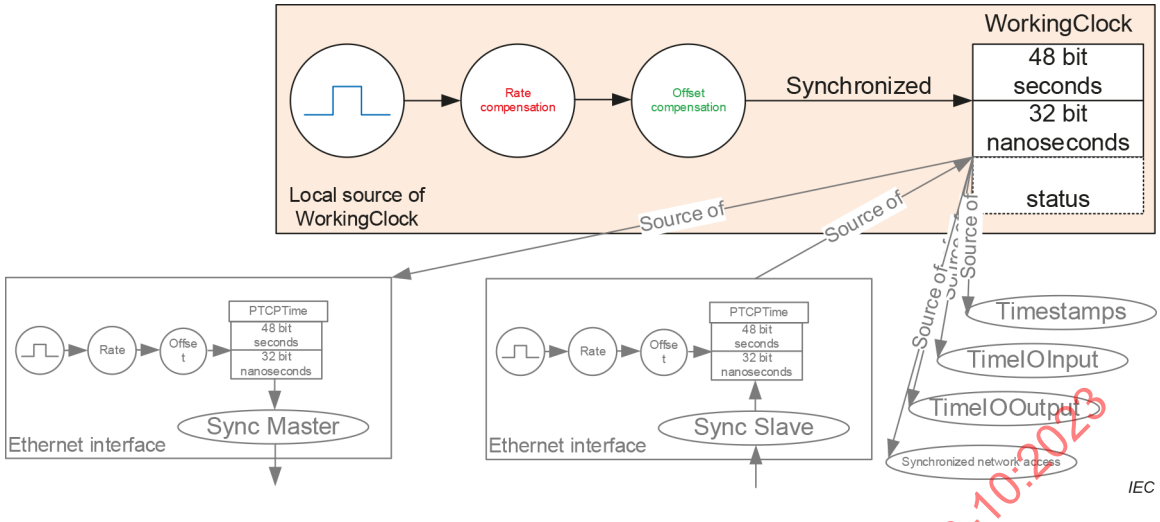


Figure 67 – WorkingClock timer model

4.5.3.2 Correspondence between WorkingClock and CycleCounter

The CycleCounter and its usage are shown in Figure 68, Figure 69, Figure 70 and Figure 74, and shall be calculated according to Formula (35), Formula (36) and Formula (37).

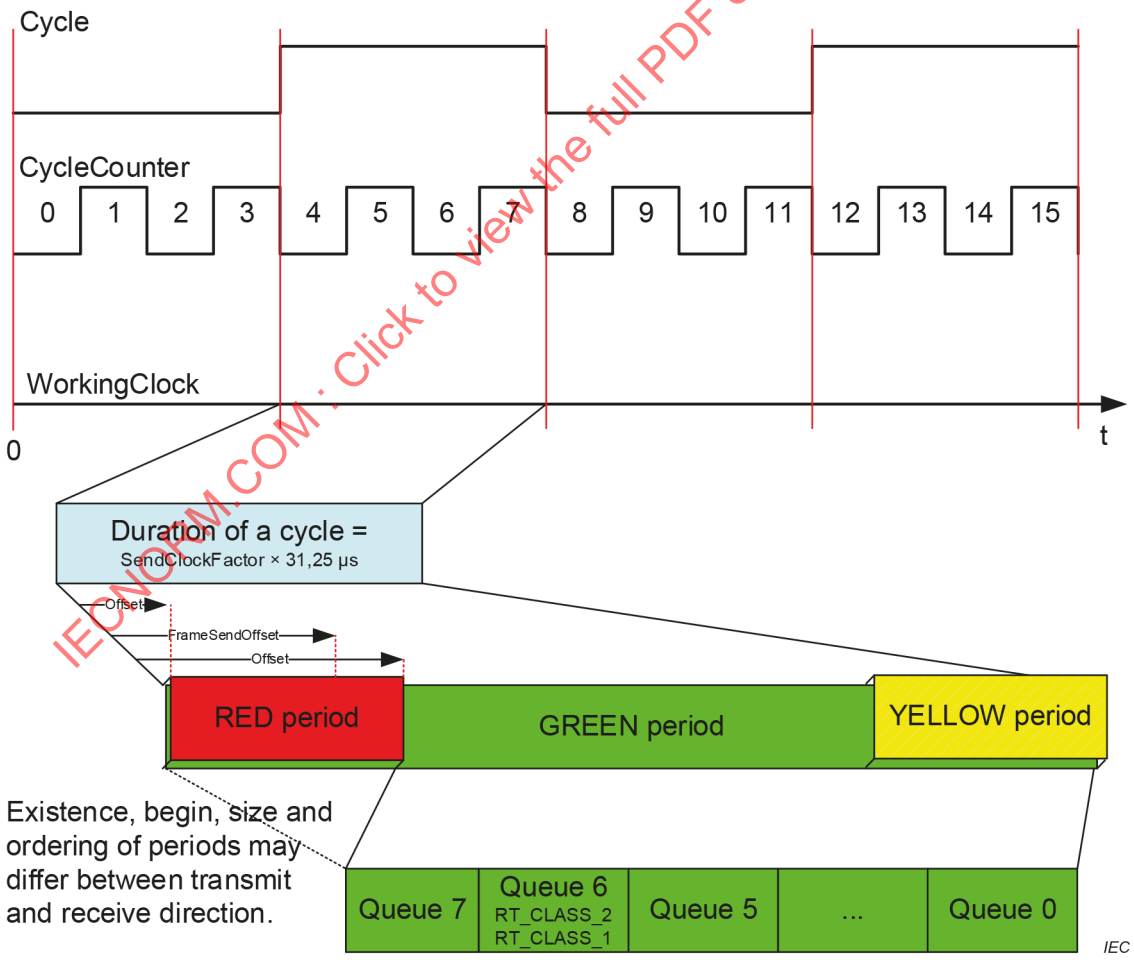


Figure 68 – Non-time-aware system – WorkingClock and CycleCounter

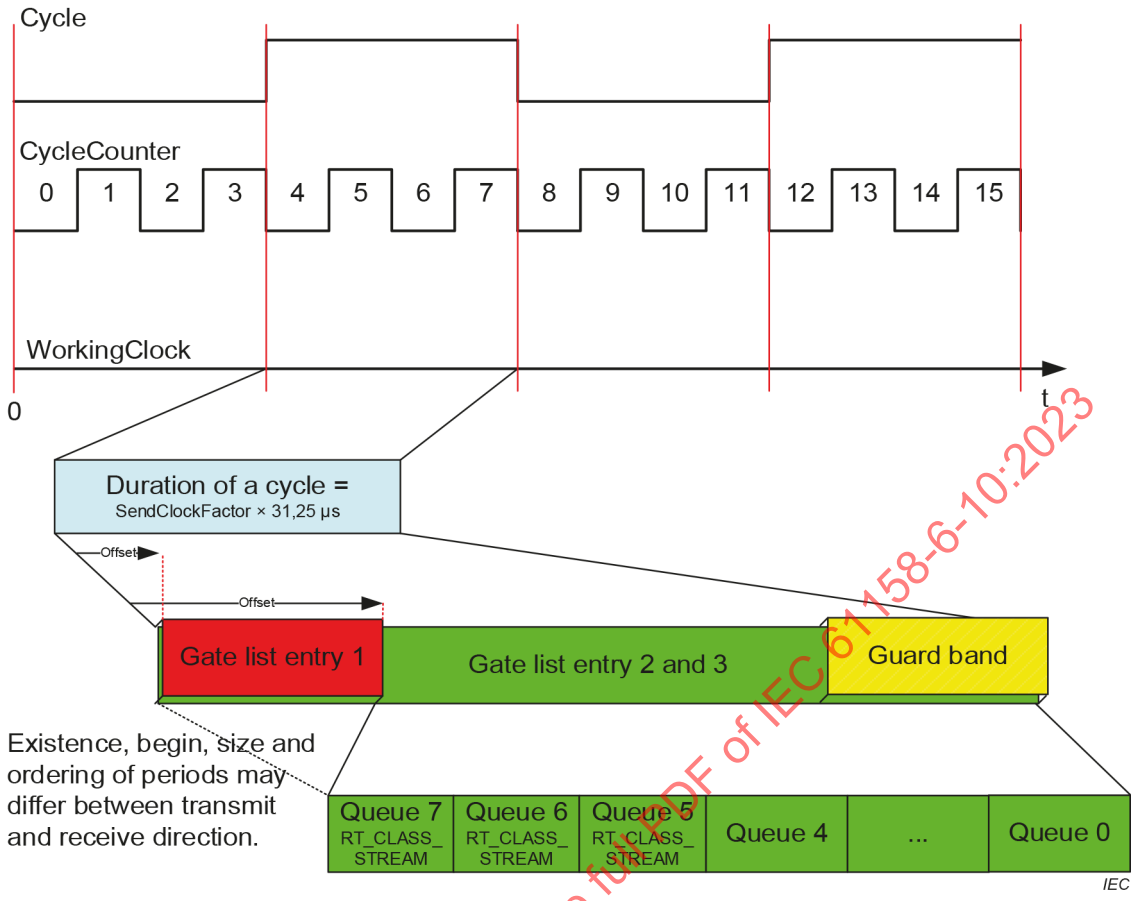
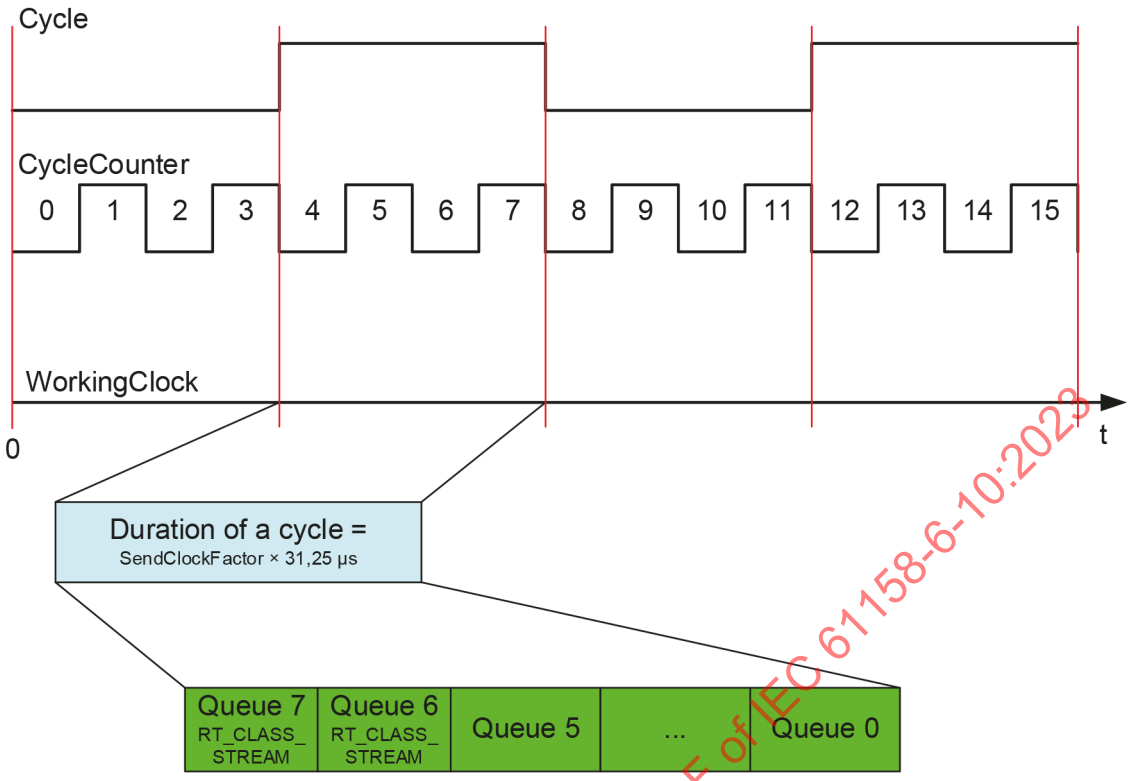


Figure 69 – Time-aware system – Queue masking – WorkingClock and CycleCounter



Existence, begin, size and ordering of periods may differ between transmit and receive direction.

IEC

Figure 70 – Time-aware system – WorkingClock and CycleCounter

$$CycleCounter = Cycle \times SendClockFactor \quad (35)$$

where

CycleCounter is the counter of cycles used for APDU_Status.CycleCounter

Cycle is the cycle

SendClockFactor is the factor of the send clock, the length of one phase

$$Cycle = WorkingClock \; DIV \; (SendClockFactor \times 31{,}25\ \mu s) \quad (36)$$

where

Cycle is the cycle

WorkingClock is the WorkingClock time

SendClockFactor is the factor of the send clock, the length of one phase

NOTE 1 DIV does an integer division and delivers the quotient.

$$\text{Offset} = \text{WorkingClock } \textit{MOD} \text{ (SendClockFactor} \times 31{,}25\ \mu\text{s)} \quad (37)$$

where

Offset	is the offset at the start of the cycle used for FrameSendOffset
WorkingClock	is the WorkingClock time
SendClockFactor	is the factor of the send clock, the length of one phase

NOTE 2 MOD does an integer division and delivers the remainder.

4.6 Media redundancy

4.6.1 Media redundancy and loop prevention

4.6.1.1 General

For media redundancy, IEC 62439-2 applies, except that the support of MRP-MIB is optional. Annex V shows the relation between the used selection of IEC 62439-2 fields and this document. All other IEC 62439-2 fields are used as defined by IEC 62439-2.

4.6.1.2 Ring

4.6.1.2.1 General

Two kinds of nodes, MRC and MRM, are defined as shown in Figure 71.

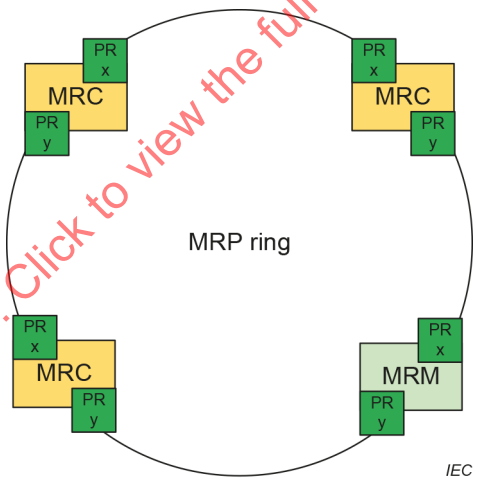


Figure 71 – Media redundancy – Ring

4.6.1.2.2 Default parameter for MRP_Role := “Media Redundancy Client”

The “MRC parameters and constraints” parameter set from IEC 62439-2 applies.

4.6.1.2.3 Default parameter for MRP_Role := “Media Redundancy Manager”

4.6.1.2.3.1 General

The “MRM parameters” parameter set for “Max. recovery time” := 200 ms from IEC 62439-2 applies.

The MRP_Prio shall be set according to Table 206.

Table 206 – Conjunction between supported MRP_Role and default MRP_Prio

Supported MRP_Role	Default MRP_Prio
Media Redundancy Manager (Auto manager negotiation) [a]	0xA000
Media Redundancy Manager [a]	0x8000
[a] Any other allowed value from Table 908 may be used as manufacturer specific default.	

4.6.1.2.3.2 "Media Redundancy Manager" with "Auto manager negotiation"

For the implementation of a MRA the definitions of IEC 62439-2 including Annex A apply.

4.6.1.2.4 Additional forwarding rules

For vendor specific applications, for example "RedApplication", which requires redundant transport in conjunction with System Redundancy, each "Media Redundancy Client" and each "Media Redundancy Manager" shall support an extended forwarding rule according to Table 207 for the MAC address shown in Table 208.

If MRP is disabled, then the default forwarding rule for multicast MAC addresses applies. Optionally, frames with the defined multicast MAC address may be discarded.

Table 207 – Extended forwarding rule

MRP_Role	Forwarding	RedApplication: Local send	RedApplication: Local receive
Media Redundancy Manager or Media Redundancy Manager (Auto manager negotiation)	No forwarding	RPort_1 and RPort_2	RedApplication exists: Local receive Without RedApplication: Drop frame
Media Redundancy Client	RedApplication exists: No forwarding Without RedApplication: Forward packets between RPort_1 and RPort_2	RPort_1 and RPort_2	RedApplication exists: Local receive Without RedApplication: Drop frame

Table 208 – Managed Multicast MAC address

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Comment
01-0E-CF	00-05-00	Managed Multicast MAC address for vendor specific usage in conjunction with MRP. A vendor specific EtherType shall be used.

4.6.1.3 Ring interconnection

4.6.1.3.1 General

Two kinds of nodes, MIC and MIM, are defined as shown in Figure 72.

The “LC-mode” as stated in IEC 62439-2 applies.

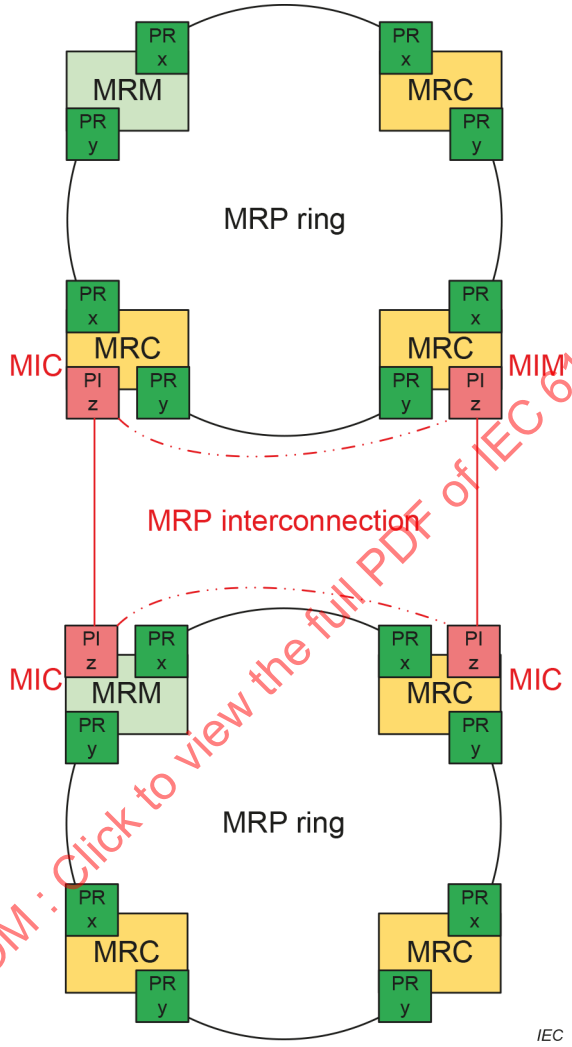


Figure 72 – Media redundancy – Interconnection

4.6.1.3.2 Default parameter for MRPIC_Role := “Media Redundancy Interconnection Client”

The “MIC parameters and constraints” parameter set from IEC 62439-2 applies.

4.6.1.3.3 Default parameter for MRPIC_Role := “Media Redundancy Interconnection Manager”

The “MIM parameters and constraints” parameter set from IEC 62439-2 applies.

4.6.1.3.4 Cross reference between MRP and MRP interconnection

The following rules shall be checked by each device offering MRP interconnection:

– A port is either MIM or MIC or MRP ring port.

– A device hosting a MIM or MIC port needs to host two ring ports at the same time.
– A device may host more than one MIM or MIC port.
– A device hosting a MIM or MIC port shall only support a single MRP instance together with a MIM or MIC port.

4.6.2 Seamless media redundancy

4.6.2.1 MRPD

For seamless media redundancy MRPD applies. This shall be done by using the PPM, CPM and the RED_RELAY.

MRPD shall only be operated in conjunction with MRP.

4.6.2.2 FRER

For seamless media redundancy together with time-aware streams FRER applies. End station FRER shall be done by using the PPM and CPM. Network FRER shall be done by using IEEE Std 802.1CB in bridge components.

FRER requires an additional protocol e.g., MSTP for loop prevention.

4.7 Real time cyclic

4.7.1 FAL syntax description

4.7.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.7.1.2 RTC APDU abstract syntax

Table 209 defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 209 shall represent the content of the DL-SDU in Table 10.

RTC-PDUs require to avoid IEEE Std 802.3 defined Padding. Thus, it needs to ensure that this IEEE Std 802.3 Padding never happens along the transmission path to ensure that the APDU_Status is always at the end of the DLPDU.

This shall be implemented independent from any VLAN or other tag stripping along the path from talker/provider to listener/consumer.

Table 209 – RTC APDU syntax

APDU name	APDU structure
RTC-PDU	C_SDU ^ CSF_SDU, [RTCPadding*] [a] [b], APDU_Status
UDP-RTC-PDU	IPHeader, UDPHeader, FrameID, RTC-PDU

[a] In case of RTC-PDU the number of padding octets shall be in the range of 0…40 depending on the DataItem size. In case of UDP-RTC-PDU the number of padding octets shall be in the range of 0…12 depending on the DataItem size.

[b] In case of RT_CLASS_3 transportation the number of padding octets is given by the engineering system.

Table 210 defines structures for substitutions of elements of the APDU structures shown in Table 209.

Table 210 – RTC substitutions

Substitution name	Structure
C_SDU [a]	{[DataItem], [GAP*]}*
DataItem	{[IOCS*], [DataObjectElement*]}*
SubstituteDataItem	{[IOCS*] [b], [SubstituteDataObjectElement*]}*
DataObjectElement	[Data*], IOPS
SubstituteDataObjectElement	[Data*], SubstituteDataValid
CSF_SDU [a]	SFCRC16, SUBFRAME*, SFEndDelimiter
SUBFRAME	SFPosition, SFDataLength, SFCycleCounter, DataStatus, C_SDU, SFCRC16
SFEndDelimiter	SFPosition (=0), SFDataLength (=0)
APDU_Status	CycleCounter, DataStatus, TransferStatus
[a] The maximum size of a C_SDU or CSF_SDU shall be 1 440 octets.	
[b] The value shall contain the same value as the field DataItem.IOCS.	

4.7.2 FAL transfer syntax

4.7.2.1 Coding section of Data-RTC-PDU specific fields

4.7.2.1.1 Overview

The fields Data and IOxS shall also be used to encode user data for all other PDU types.

4.7.2.1.2 Coding of the field CycleCounter

This field shall be coded as data type Unsigned16. One increment represents a time value of 31,25 µs for RTC-PDU and 1 ms for UDP-RTC-PDU. Each RTC-PDU or UDP-RTC-PDU contains its CycleCounter created according to Figure 74 by the sender.

The receiver of the RTC-PDU or UDP-RTC-PDU shall use this field to detect repetitions, loss of frames and timeliness of data. For this reason, the CycleCounter value range shall be divided in the ratio 15/16 for valid and 1/16 for invalid as shown in Figure 73.

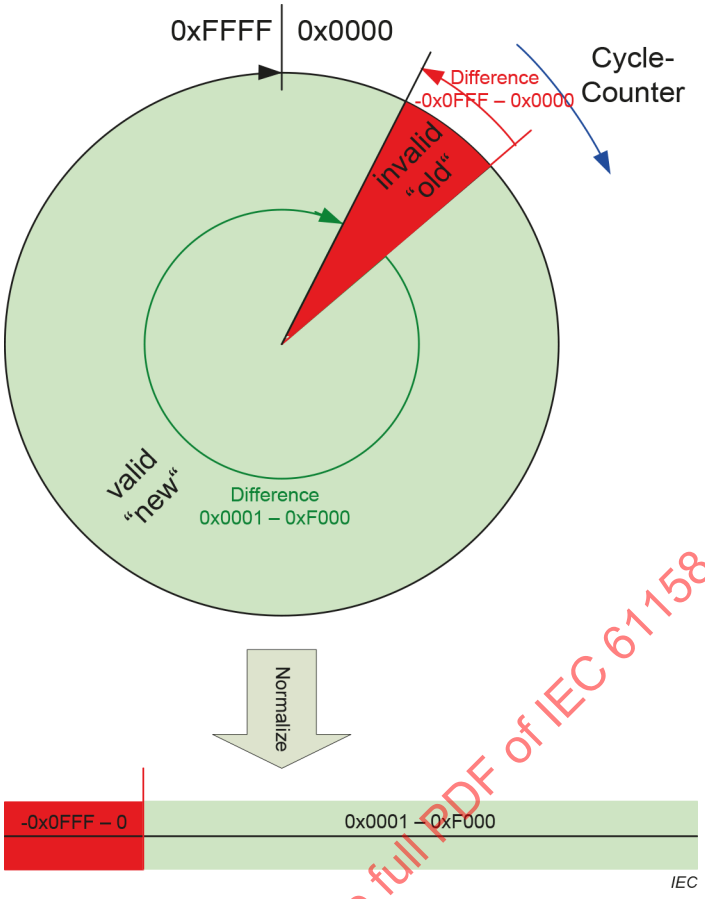


Figure 73 – CycleCounter value range

NOTE The normalization can be done using the following formula:
CycleCounter(Normalized) := ((((CycleCounter(NEW) + (0x10000 – CycleCounter(STORED)) – 1) & 0xFFFF) – 0xF000) × (-1))

Table 211 defines a maximum time window between the old and the new frame and the action of the consumer according to the difference between the CycleCounter of the last, identified as, valid and thus stored RTC-PDU or UDP-RTC-PDU and the value of the currently received PDU.

Table 211 – CycleCounter Difference

Value (hexadecimal)	Time range RTC-PDU [s]	Time range UDP-RTC-PDU [s]	Meaning	Use
-0x0FFF – 0x0000	0,128	4,096	Red sector according to Figure 73 1/16 of the CycleCounter range The consumer votes the received frame as older than the stored frame	Frame is dropped
0x0001 – 0xF000	1,92	61,44	Green sector according to Figure 73 15/16 of the CycleCounter range The consumer votes the received frame as newer than the stored frame	Frame is processed

If MRPD or end station FRER is used, the CycleCounter of the adjunctive frames shall be set identical by the sender.

A device shall maintain a local 64 bit counter with an increment of 31,25 µs as shown in Figure 74.

The bits 0 to 14 shall be used identically for the CycleCounter of the RTC-PDUs. The bit 15 of the 16 bit CycleCounter of the RTC-PDUs may also be identical to bit 15 of the 64 bit local counter.

The bits 5 to 19 shall be used identically as bit 0 to 14 for the CycleCounter of the UDP-RTC-PDUs. The bit 15 of the 16 bit CycleCounter of the UDP-RTC-PDUs may also be identical to bit 20 of the 64 bit local counter.

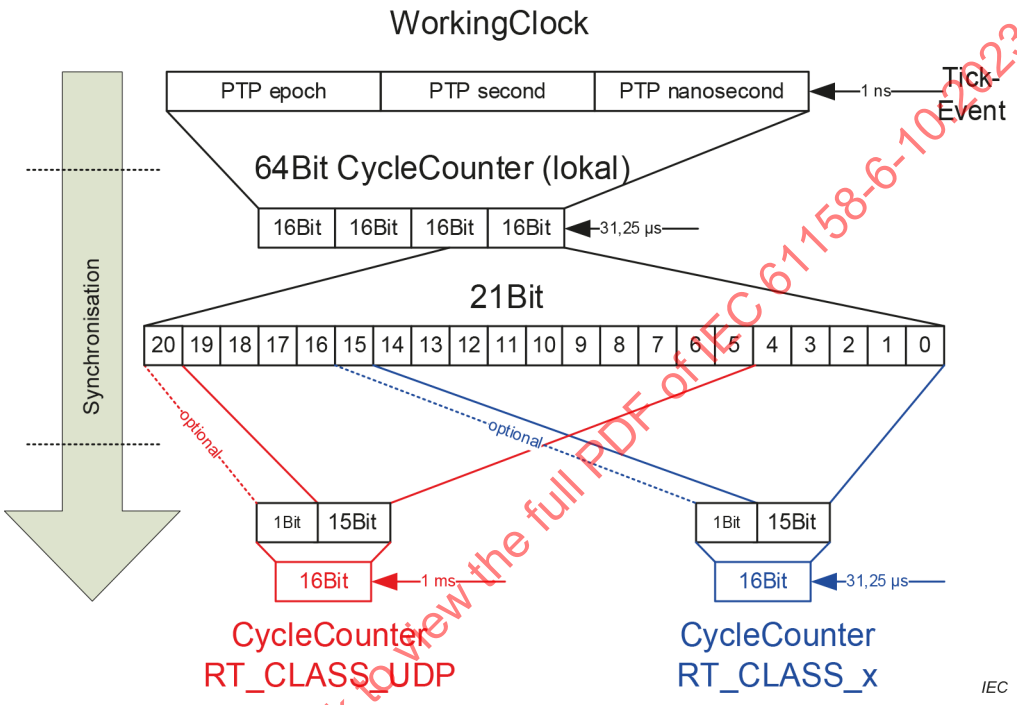


Figure 74 – Structure of the CycleCounter

In case of a global synchronization of the nodes, the bit 15 of the CycleCounter base for RTC-PDUs and the bit 20 of the CycleCounter base for UDP-RTC-PDUs shall be set according to Figure 74 and Figure 75 to avoid a false “older” detection by the receiver.

The CycleCounter shall be set due to synchronization according to the model shown in Figure 75.

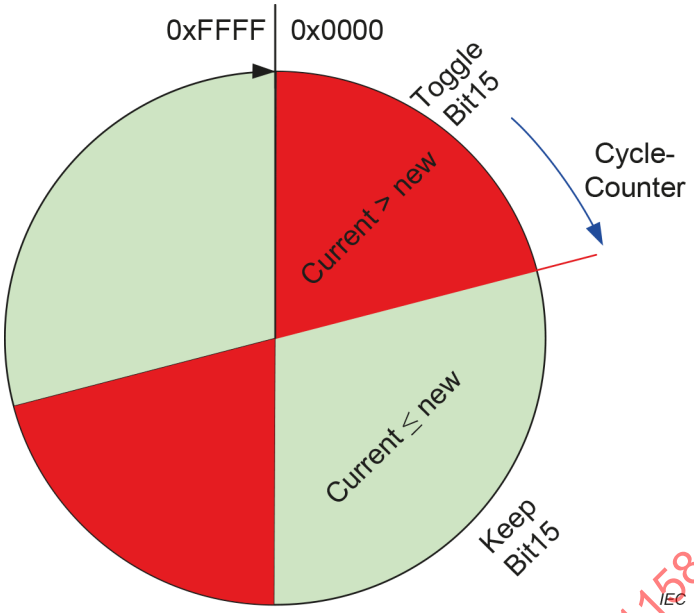


Figure 75 – Optimized CycleCounter setting

CycleCounter(current)
This means the value of the CycleCounter used by the sender before executing the synchronization

CycleCounter(next)
This means the value of the CycleCounter needed by the sender according to the synchronization information

if CycleCounter(current)[0..14] <= CycleCounter(next)[0..14]

then /* current ≤ next */

SET CycleCounter[0..14] with CycleCounter(next)[0..14] and keep CycleCounter[15]

else /* current > next */

SET CycleCounter[0..14] with CycleCounter(next)[0..14] and toggle CycleCounter[15]

4.7.2.1.3 Coding of the field DataStatus

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0: DataStatus.State

This field shall be coded with the values according to Table 212 and evaluated when the AR is in state InDATA.

Table 212 – DataStatus.State

Value (hexadecimal)	Meaning
0x00	IOCR state is backup
0x01	IOCR state is primary

Bit 1: DataStatus.Redundancy

This field shall be coded with the values according to Table 213 and Table 214, and evaluated when the AR is in state InDATA.

NOTE An IO controller sets this flag independently from the ARType to the Default value.

Table 213 – DataStatus.Redundancy in conjunction with DataStatus.State==Backup

Value (hexadecimal)	Meaning
0x00	Default One primary AR of a given AR-set is present
0x01	No primary AR of a given AR-set is present [a]
[a] Only issued if the RDHT is started (after the first Backup -> Primary transition).	

Table 214 – DataStatus.Redundancy in conjunction with DataStatus.State==Primary

Value (hexadecimal)	Meaning
0x00	Default The ARstate from the IO device point of view is Primary
0x01	The ARstate from the IO device point of view is Backup

Bit 2: DataStatus.DataValid
This field shall be coded with the values according to Table 215.

Table 215 – DataStatus.DataValid

Value (hexadecimal)	Meaning
0x00	DataItem invalid
0x01	DataItem valid

For a Subframe the DataStatus.DataValid shall be set to “DataItem valid”.

Bit 3: DataStatus.Reserved_1
This field shall be set to zero but not checked by the receiver.

Bit 4: DataStatus.ProviderState
This field shall be coded with the values according to Table 216.

Table 216 – DataStatus.ProviderState

Value (hexadecimal)	Meaning
0x00	Stop
0x01	Run

Bit 5: DataStatus.StationProblemIndicator
This field shall be coded with the values according to Table 217.

Table 217 – DataStatus.StationProblemIndicator

Value (hexadecimal)	Meaning	Usage
0x00	Problem detected	Fault detected The Diagnosis ASE contains at least one DiagnosisData with ChannelProperties.Maintenance(=Fault) and ChannelProperties.Specifier(=appear) for the AR containing this CR.
0x01	Normal Operation	Fault free The Diagnosis ASE contains no DiagnosisData with ChannelProperties.Maintenance(=Fault) and ChannelProperties.Specifier(=appear) for the AR containing this CR.

Bit 6: DataStatus.Reserved_2
This field shall be set according to 3.4.2.2.

Bit 7: DataStatus.Ignore
This field shall be coded with the values according to Table 218 and Table 219.

Table 218 – DataStatus.Ignore of a frame

Value (hexadecimal)	Meaning
0x00	Evaluate the DataStatus
0x01	Ignore the DataStatus Shall only be set for a frame containing sub frames. The CPM may evaluate the DataStatus.

Table 219 – DataStatus.Ignore of a sub frame

Value (hexadecimal)	Meaning
0x00	Evaluate the sub frame
0x01	Ignore the sub frame; it is a surrogate sub frame. The CPM treats this Subframe as invalid.

4.7.2.1.4 Coding of the field TransferStatus

This field shall be coded as data type Unsigned8. This field shall be set to zero for RT_CLASS_UDP, RT_CLASS_1 and RT_CLASS_2 frames. For RT_CLASS_3 the value shall be set according to Table 220.

Table 220 – TransferStatus for RT_CLASS_3

Bit position	Name	Value	Meaning
0	AlignmentOrFrameChecksumError	0	No frame checksum error or alignment error
		1	A frame checksum error or alignment error and no MAC receive buffer overflow has occurred
1	WrongLengthError	0	No length error
		1	A length error has occurred
2	MACReceiveBufferOverflow	0	No MAC receive buffer overflow
		1	A MAC receive buffer overflow has occurred
3	RT_CLASS_3 Error	0	No RT_CLASS_3 error
		1	RT_CLASS_3 error For example the frame has not been received in time or does not have the expected frame type or does not have the expected frame ID or does not have the expected Source MAC address (this is optionally checked) or in case of an underrun, for example the bridge received a header, but the data has not arrived on time
Other	—	Reserved	Reserved

4.7.2.1.5 Coding section related to distributed I/O

4.7.2.1.5.1 Coding of the field Data

The data types defined in IEC 61158-5-10, Clause 5 shall be used for Data.

4.7.2.1.5.2 Coding section related to Subframe

4.7.2.1.5.2.1 Coding of the field SFCRC16

This field shall be coded as data type Unsigned16.

The SFCRC16 is generated by Formula (38) as a 16 Bit-polynomial.

$$SFCRC16 = 1 + x^2 + x^{15} + x^{16} \quad (38)$$

where

x is the content of the octet

SFCRC16 16 Bit-polynomial

Each SFCRC16 starts with value 0 and calculates all octets beginning with first octet of destination MAC address up to the SFCRC16.

The 16 bit value representation in a frame is little endian (which ensures that the value of CRC is 0 after the SFCRC16 octets of the previous Subframe).

NOTE The design can be checked using www.easics.be/webtools/crctool.

The generation of the first SFCRC16 starts with the first octet of the destination MAC address, ignores the VLAN tag (and other tags) and ends with the last octet of the FrameID.

The generation of the Subframe SFCRC16 starts with the first octet of the SFPosition and ends with the last octet of the field C_SDU, as shown in Figure 76.

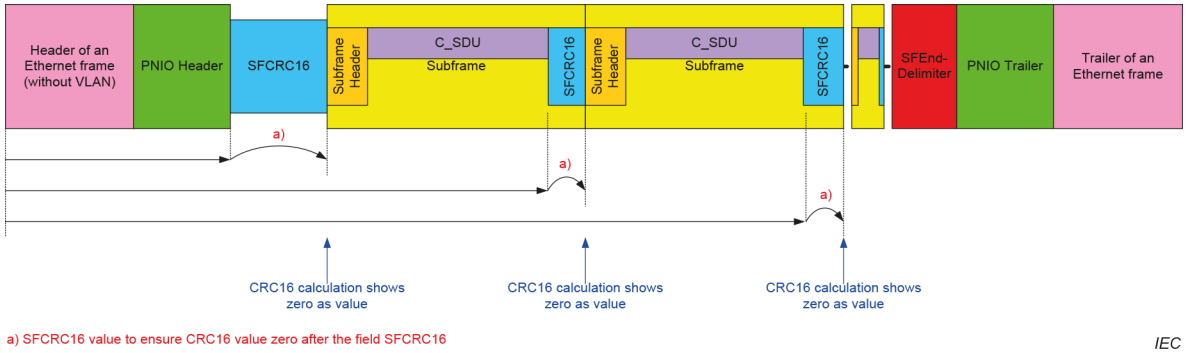


Figure 76 – SFCRC16 generation rule

4.7.2.1.5.2.2 Coding of the field SFPosition

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0 – 6: SFPosition.Position

This field shall be coded with the values according to Table 221.

Table 221 – SFPosition.Position

Value (hexadecimal)	Meaning
0x00	End of Subframe coding
0x01 – 0x3F	Usable for packetizing and unpacking of Subframes during transport and end to end
0x40 – 0x7F	Usable for packetizing and unpacking of Subframes end to end

NOTE End to end means that the source of data creates and concatenates the Subframes of a frame. The conveyance is done without any change to the frame. The sink of the data unpacks the Subframes and uses the data.

Bit 7: SFPosition.Reserved

This field shall be coded with the values according to Table 222.

Table 222 – SFPosition.Reserved

Value (hexadecimal)	Meaning
0x00	Reserved

4.7.2.1.5.2.3 Coding of the field SFDataLength

This field shall be coded as data type Unsigned8 with the values according to Table 223.

Table 223 – SFDataLength

Value (hexadecimal)	Meaning
0x00	End of Subframe coding
0x01 – 0xFF	Number of octets of the C_SDU

4.7.2.1.5.2.4 Coding of the field SFCycleCounter

This field shall be coded as data type Unsigned8. Each Subframe contains its SFCycleCounter incremented by one for each transmission from the PPM.

The receiver of the Subframe shall use this field to detect repetitions, loss of Subframes and timeliness of data. For this reason, the SFCycleCounter value range shall be divided in the ratio 15/16 for valid and 1/16 for invalid as shown in Figure 77.

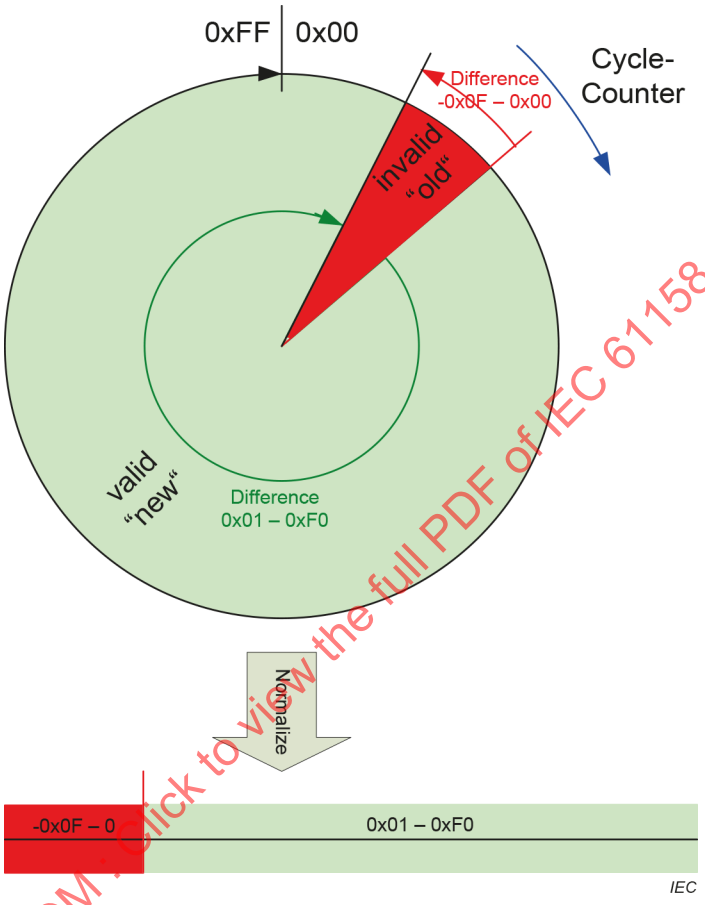


Figure 77 – SFCycleCounter value range

Table 224 defines the maximum time window between the old and the new SFCycleCounter and the action of the consumer according to the difference between the SFCycleCounter of the last received Subframe and the value of the currently received Subframe.

Table 224 – SFCycleCounter Difference

Value (hexadecimal)	Time range RTC-PDU [s]	Time range UDP-RTC-PDU [s]	Meaning	Use
-0x0F– 0x00	—	—	Red sector according to Figure 77 1/16 of the SFCycleCounter range The consumer votes the received Subframe as older than the stored Subframe	Subframe is dropped
0x01 – 0xF0	—	—	Green sector according to Figure 77 15/16 of the SFCycleCounter range The consumer votes the received Subframe as newer than the stored Subframe	Subframe is processed

NOTE The same SFCycleCounter value, which means the difference is zero, is assigned to “older”.

If MRPD is used, the SFCycleCounter of the adjunctive subframes shall be set identical by the sender.

4.7.2.1.5.3 Coding section related to IOxS (IOPS, IOCS, SubstituteDataValid)

4.7.2.1.5.3.1 Coding of the field IOPS

The coding of these fields shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0: IOxS.Extension
This field shall be coded with the values according to Table 225.

Table 225 – IOxS.Extension

Value (hexadecimal)	Meaning
0x00	No IOxS octet follows
0x01	One more IOxS octet follows

Bit 1 to 4: IOxS.Reserved
This field shall be set according to 3.4.2.2.

Bit 5 to 6: IOxS.Instance
This field shall be coded with the values according to Table 226 if the IOxS.DataState bit is set to BAD. If this bit is set to GOOD, the IOxS instance bits are “don’t care” and shall be set to zero.

Table 226 – IOxS.Instance

Value (hexadecimal)	Meaning
0x00	Detected by subslot
0x01	Detected by slot
0x02	Detected by IO device
0x03	Detected by IO controller

Bit 7: IOxS.DataState
This field shall be coded with the values according to Table 227.

Table 227 – IOxS.DataState

Value (hexadecimal)	Meaning
0x00	BAD Data field can contain invalid user data – the receiver shall use its pre-configured values (zero, last valid value, or default value) instead of transmitted data field values SubstituteDataValid = FALSE
0x01	GOOD Data field shall contain valid user data SubstituteDataValid = TRUE

4.7.2.1.5.3.2 Coding of the field IOCS

The coding of these fields shall be according to 4.7.2.1.5.3.1.

4.7.2.1.5.3.3 Coding of the field SubstituteDataValid

The coding of these fields shall be according to 4.7.2.1.5.3.1.

4.7.3 FAL Service Protocol Machines

There is no FAL Service Protocol Machine (FSPM) defined for this Protocol.

4.7.4 Application Relationship Protocol Machines

4.7.4.1 Buffer Lifetime Model

4.7.4.1.1 General

This model shows a way to support the maximum number of controlled devices together with minimum SendClocks in Multichip and/or Multicore environment.

PCIe and the PCIe feature PTM are just shown as an example for a synchronized Multichip and/or Multicore connection.

4.7.4.1.2 Principle

The Buffer Lifetime Model, as shown in Figure 78, Figure 79 and Figure 80, provides non-blocking access to buffer with full consistency, even with a Multichip and/or Multicore environment in between, if occupancy follows agreed rules.

Rules:

- Buffer pools and descriptor tables
  - are allocated by the Host
  - Addresses and structure of both, pools and descriptors, are known by Host and Interface
  - Expected Buffer Lifetime is known by Host and Interface
  - Cache line alignment is used for both buffers and descriptor Table entries

- Buffers are managed by the source of data
  - Allocated and filled with actual data
  - Reference updated in descriptor table
  - Previous buffer protected (unchanged) for the agreed Buffer Lifetime
- Buffers are used for an agreed time interval from the sink of data
  e.g. access the data of a buffer for an interval of e.g. 5 µs maximum
  - Reference fetched from the descriptor table
  - Buffer behind reference copied to a local buffer staying below the agreed Buffer Lifetime

The source of information manages the buffers, the assigned buffer pointers, and the descriptor Table as shown in Figure 78 to allow the flow shown in Figure 79.

PPM:
The Host or Application is source of data and thus manages the buffers.

CPM:
Network/Ethernet interface is source of data and thus manages the buffers.

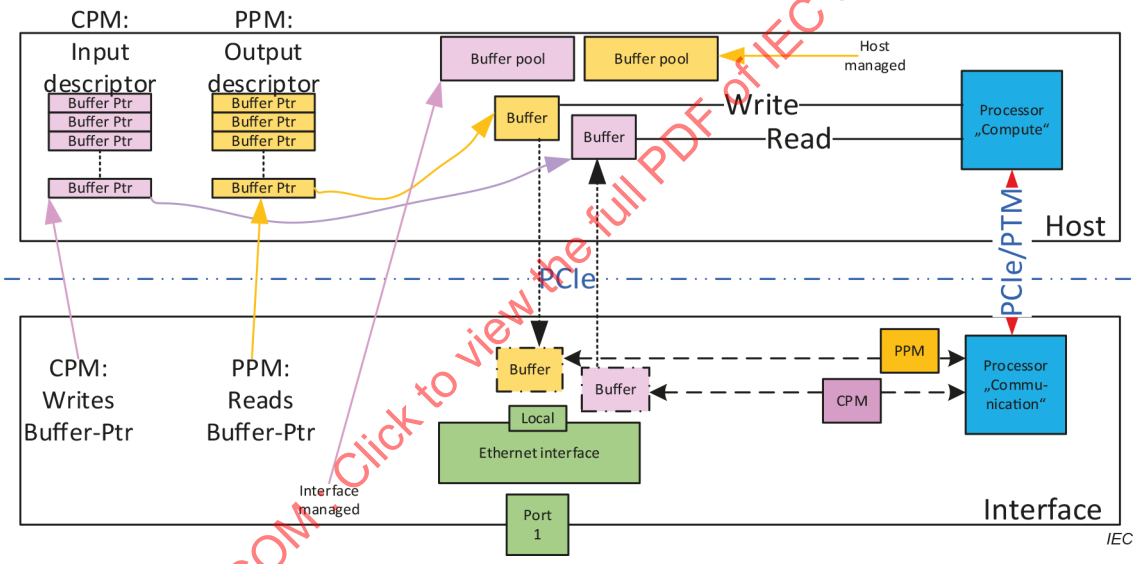


Figure 78 – Overview Buffer Lifetime Model

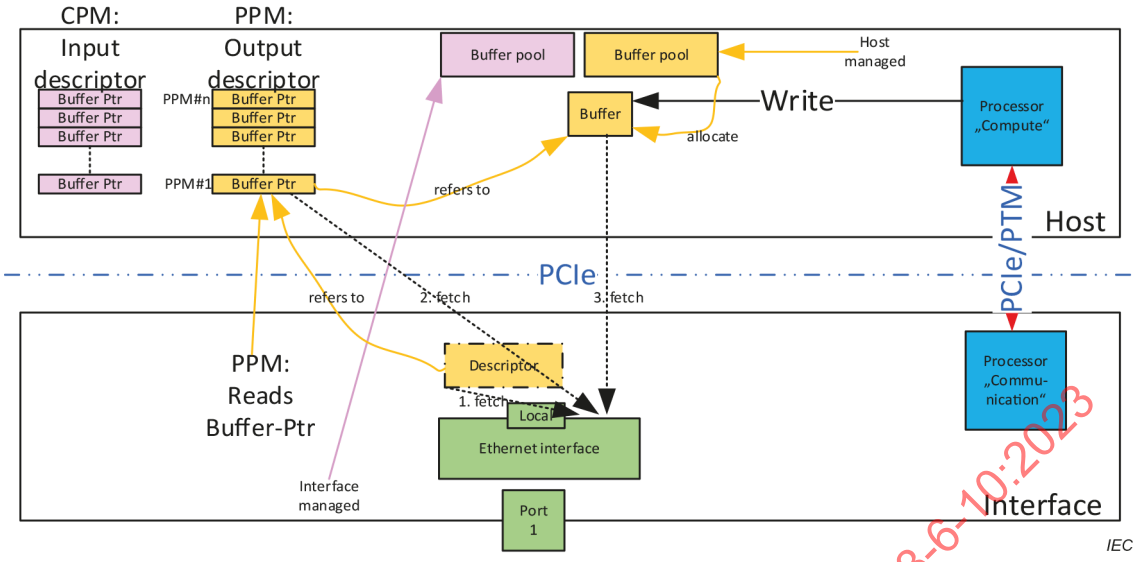


Figure 79 – PPM Flow Model

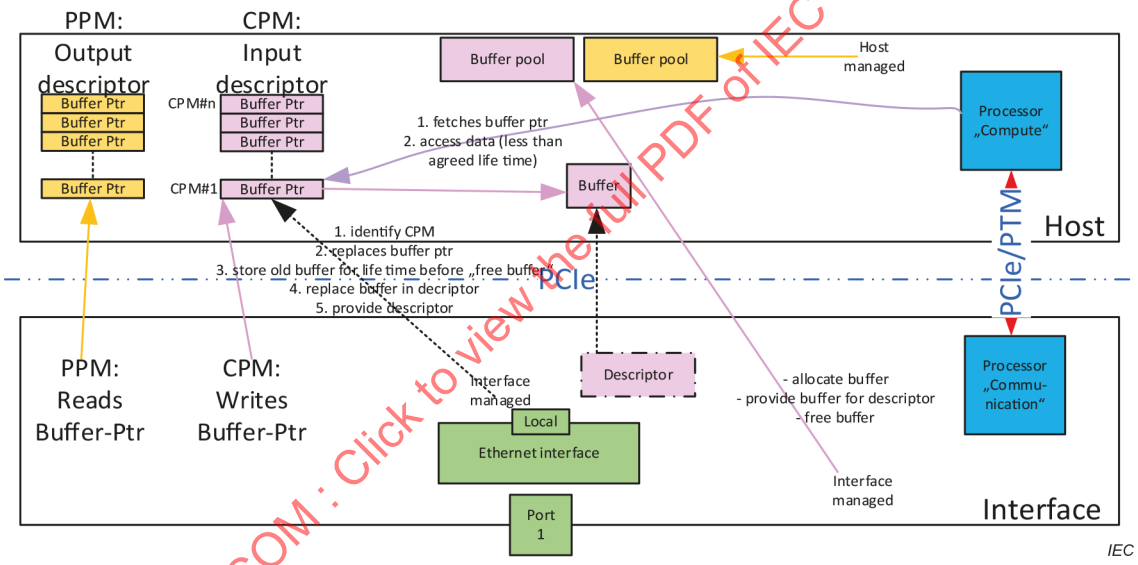


Figure 80 – CPM Flow Model

Data source will always behave in the following way:

a) Allocate buffer from pool
   “Get a new buffer”
b) Fill allocated buffer with data
   “Prepare new data”
c) Update buffer pointer in the descriptor table
   “Provide new/actual data”
d) Keep previous buffer unchanged for the agreed amount of time
   “Ensure consistency by protecting previous buffer”
e) Free previous buffer when time expires
   “Put buffer to the pool”

Data sink will always behave in the following way:

a) Get buffer pointer from descriptor table

   “Get reference to new/actual data”

b) Copy buffer into locally owned buffer

   “Get new/actual data”

c) Check buffer Header and Trailer using the locally owned buffer to detect any break of the agreement

   “Check for break of Buffer Lifetime agreement”

d) Make sure that a) + b) need less than the agreed amount of time

   “Behave well”

4.7.4.1.3 Robustness

To detect application handling errors inside the Buffer Lifetime Model, a pattern needs to be added by the source at the head and tail of the buffer and checked by the sink.

– CPM controlled buffers:

  The application reads the pattern at the head when using the buffer and compares the value with the tail when done.

– Application controlled buffers:

  The PPM reads the pattern at the head when using the buffer and compares the value with the tail when done.

Such a check supports the detection of handling and runtime errors.

4.7.4.2 Provider protocol machine

4.7.4.2.1 General

Figure 81 and Figure 82 show the basic structure of the Provider Protocol Machine (PPM). Table 228 shows the APDU_Status of a PPM with subframe structure.

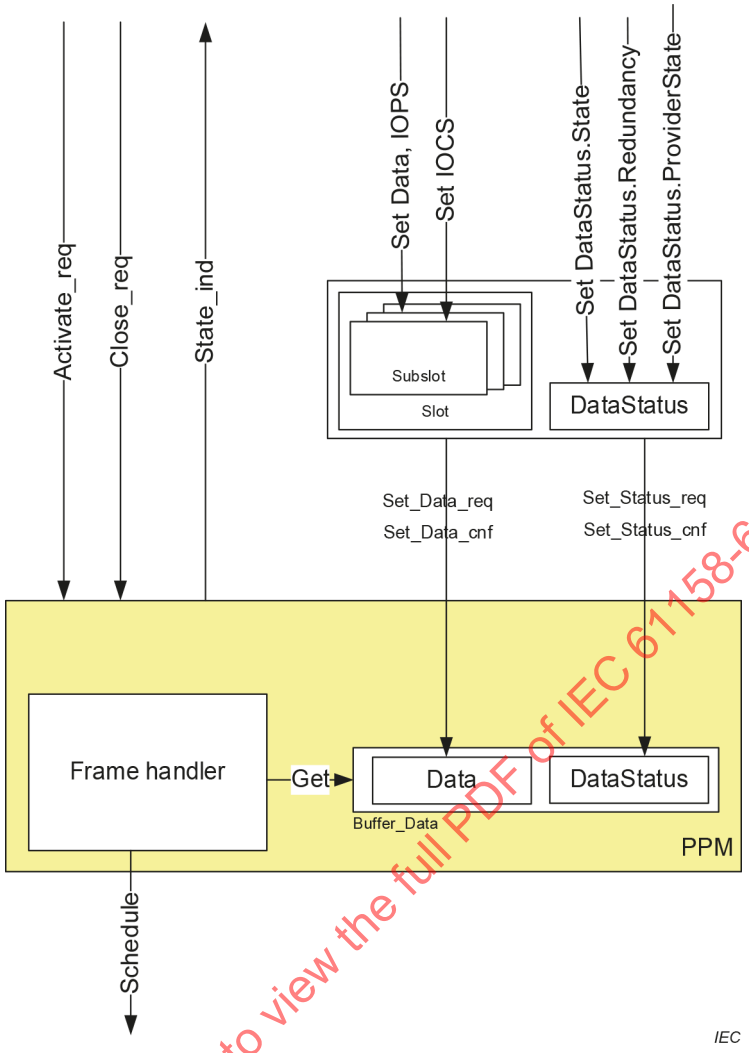


Figure 81 – Basic structure of a PPM with frame structure

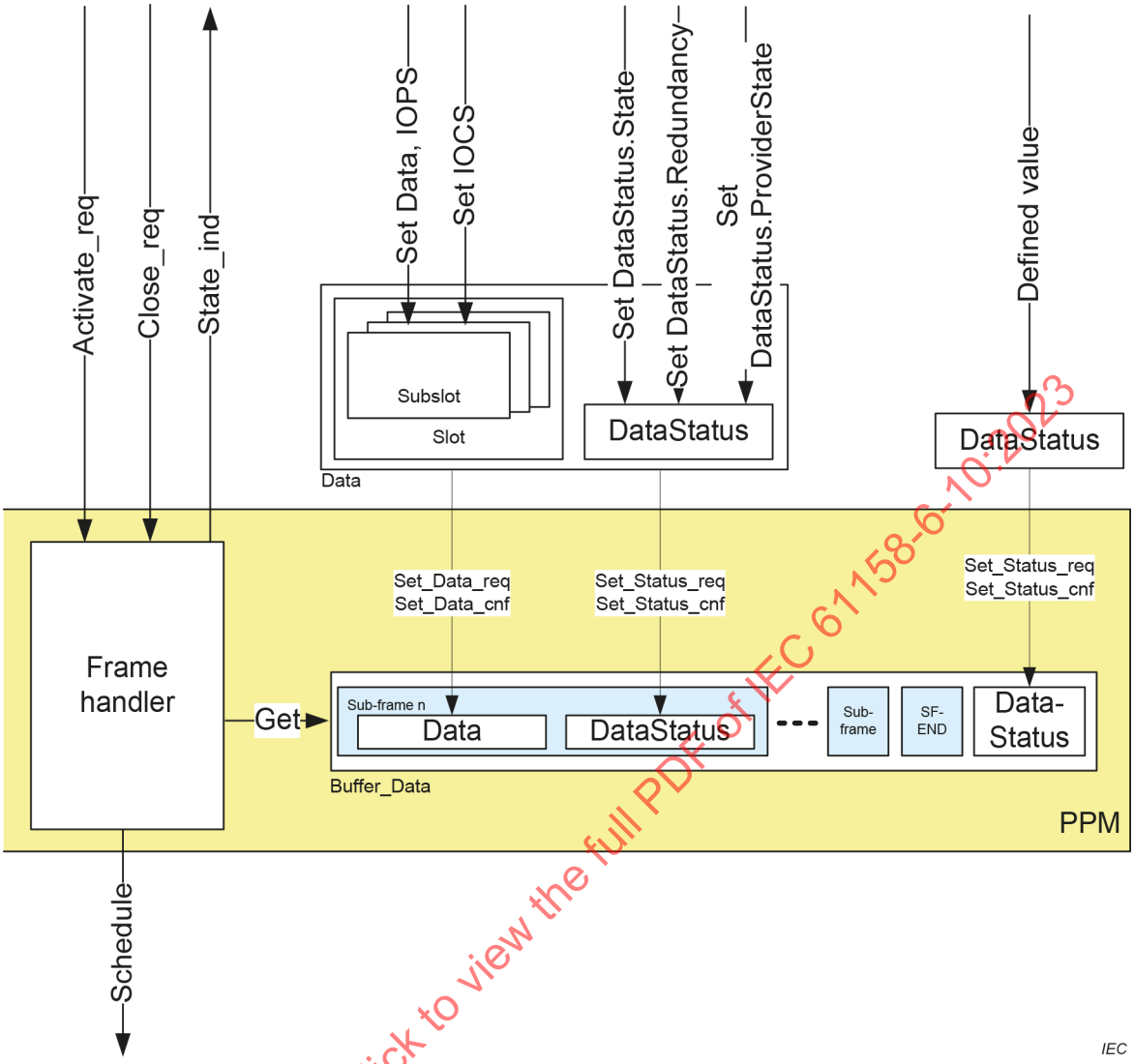


Figure 82 – Basic structure of a PPM with subframe structure

Table 228 – APDU_Status of a PPM with subframe structure

Name	Function/Meaning
CycleCounter	Shall contain the “actual local value” derived from the PTCP synchronized SendClock.
TransferStatus	Shall be set to zero.
DataStatus	DataStatus.State:=1, DataStatus.Redundancy:=0, DataStatus.DataValid:=1, DataStatus.ProviderState:=1, DataStatus.StationProblemIndicator:=1, DataStatus.Ignore:=1

4.7.4.2.2 Primitive definitions

4.7.4.2.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by PPM are described in the service definition and shown in Table 229.

Table 229 – Remote primitives issued or received by PPM

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.7.4.2.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by PPM are described in the service definition and shown in Table 230.

Table 230 – Local primitives issued or received by PPM

Primitive	Source	Destination	Associated parameters	Functions
PPM_Activate_cnf (-)	PPM	CMSU CTLSU	CREP, ERRCLS, ERRCODE	—
PPM_Activate_cnf (+)	PPM	CMSU CTLSU	CREP	—
PPM_Activate_req	CMSU CTLSU	PPM	CREP, DA, SA, FrameID, Prio, TxOption, ReductionRatio, Phase, Sequence, Default_Values, Default_Status	—
PPM_Close_cnf (-)	PPM	CMSU CTLSU	CREP, ERRCLS, ERRCODE	—
PPM_Close_cnf (+)	PPM	CMSU CTLSU	CREP	—
PPM_Close_req	CMSU CTLSU	PPM	CREP	—
PPM_Error_ind	PPM	CMSU CTLSU	CREP, ERRCLS, ERRCODE	—
PPM_Set_Data_cnf (-)	PPM	FSPMDEV FSPMCTL	CREP, ERRCLS, ERRCODE	—
PPM_Set_Data_cnf (+)	PPM	FSPMDEV FSPMCTL	CREP	—
PPM_Set_Data_req	FSPMDEV FSPMCTL	PPM	CREP, Data	—
PPM_Set_Status_cnf (-)	PPM	FSPMDEV FSPMCTL	CREP, ERRCLS, ERRCODE	—
PPM_Set_Status_cnf (+)	PPM	FSPMDEV FSPMCTL	CREP	—
PPM_Set_Status_req	FSPMDEV FSPMCTL	PPM	CREP, Status	—
PPM_State_ind	PPM	local matter	CREP, State	—
Scheduler_add_cnf (-)	SCHEDULER	PPM	CREP	—
Scheduler_add_cnf (+)	SCHEDULER	PPM	CREP	—

Primitive	Source	Destination	Associated parameters	Functions
SCHEDULER_add_req	PPM	SCHEDULER	CREP, ReductionRatio, Phase, Sequence, TxOption	—
Scheduler_remove_cnf (-)	SCHEDULER	PPM	CREP	—
Scheduler_remove_cnf (+)	SCHEDULER	PPM	CREP	—
SCHEDULER_remove_req	PPM	SCHEDULER	CREP	—
Scheduler_transmit_ind	SCHEDULER	PPM	CREP	—

4.7.4.2.3 State transition diagram

The state transition diagram of the PPM is shown in Figure 83.

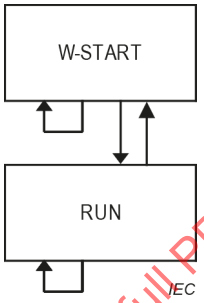


Figure 83 – State transition diagram of PPM

States of the PPM are:

W-START Waiting for initialization and add the frame to the SCHEDULER

RUN Running

4.7.4.2.4 State machine description

The W-START state indicates that an initialization is needed. The Activate service sets the machine to the RUN state waiting for Provider Data service requests and Time events. Receiving the confirmation will return the machine to the RUN state. An error or a Close service request is needed to re-enter the W-START state.

- In case of NonTimeAware systems:
  - Each PPM handles up to two frames with one FrameID.
  - MRPD is used:
    Each PPM handles up to two frames with one or two FrameIDs. For RTC3 compatibility two frames with the same FrameID and for MRPD two frames with two FrameIDs (FrameID and FrameID+1) are handled.
- In case of TimeAware systems:
  - Each PPM handles up to two frames with one FrameID.
  - MRPD is used:
    Each PPM handles up to two frames with one FrameID. For seamless redundancy (MRPD) two frames with the same FrameID, the same DestinationAddress and two different VLANs are handled.

– End station FRER is used:
Each PPM handles up to two frames with one FrameID. For seamless redundancy (FRER) two frames with the same FrameID, the same DestinationAddress, two different VLANs and FRER header are handled.

It is used to setup the frames which are transmitted by the Scheduler. The content (Buffer_Data) contains the local data if "frame structure" is used or the local data and the stored subframes (DFP_STORAGE) if "subframe structure" is used.

The Buffer_Data is fetched first from the INBOUND DFP frame or if this frame is too late from the DFP_STORAGE. If the concurrently received and transmitted subframe is detected as invalid, the frame should be shortened according to the rules of the IEEE Std 802.3 at the transmitting port.

4.7.4.2.5 PPM state table

Table 231 contains the description of the PPM state table.

Table 231 – PPM state table

#	Current State	Event /Condition =>Action	Next State
1	W-START	**PPM_Activate_req ()** => FirstTransmit := FALSE Buffer_Data := Default_Values Buffer_Status := Default_Status SCHEDULER_add.req () PPM_Activate_cnf (+)	RUN
2	W-START	**PPM_Set_Data_req (Data)** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Data_cnf (-)	W-START
3	W-START	**PPM_Set_Status_req (Status)** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Status_cnf (-)	W-START
4	W-START	**PPM_Close_req ()** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Close_cnf (-)	W-START
5	W-START	**Scheduler_transmit.ind ()** => ignore	W-START
6	W-START	**Scheduler_add.cnf (+)** => ignore	W-START
7	W-START	**Scheduler_remove.cnf (+)** => ignore	W-START
8	W-START	**Scheduler_add.cnf (-)** => ignore	W-START
9	W-START	**Scheduler_remove.cnf (-)** => ignore	W-START

#	Current State	Event /Condition =>Action	Next State
10	RUN	**PPM_Activate_req ()** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Activate_cnf (-)	RUN
11	RUN	**PPM_Set_Data_req (Data)** => Buffer_Data := Data PPM_Set_Data_cnf (+)	RUN
12	RUN	**PPM_Set_Status_req (Status)** => Buffer_Status := Status PPM_Set_Status_cnf (+)	RUN
13	RUN	**PPM_Close_req ()** => SCHEDULER_remove.req () PPM_Close_cnf (+)	W-START
14	RUN	**Scheduler_transmit.ind ()** /FirstTransmit == FALSE => FirstTransmit := TRUE PPM_State_ind (Start)	RUN
15	RUN	**Scheduler_transmit.ind ()** /FirstTransmit == TRUE => ignore	RUN
16	RUN	**Scheduler_add.cnf (+)** => ignore	RUN
17	RUN	**Scheduler_remove.cnf (+)** => ignore	RUN
18	RUN	**Scheduler_add.cnf (-)** => ERRCLS := CTXT ERRCODE := INVALID PPM_State_ind (Error) PPM_Error_ind (Error)	W-START
19	RUN	**Scheduler_remove.cnf (-)** => ignore	RUN

4.7.4.2.6 Functions, Macros, Timers and Variables

Table 232 contains the functions, macros, timers and variables used by the PPM and their arguments and their descriptions.

Table 232 – Functions, Macros, Timers and Variables used by the PPM

Name	Type	Function/Meaning
StartTimer	Function	This local function starts or restarts a timer.
StopTimer	Function	This local function stops a timer.
TimerExpired	Function	This local function is issued if a timer expires.
FirstTransmit	Variable	This local variable contains the information whether the frame was conveyed at least one time.
Buffer_Data	Variable	This local variable contains the storage of the data (encoded as DataItem*) dedicated for conveyance of the next Data-RTC-PDU or UDP-RTC-PDU.
Buffer_Status	Variable	This local variable contains the storage of the status (encoded as APDU_Status.DataStatus) dedicated for conveyance of the next Data-RTC-PDU or UDP-RTC-PDU.
NOTE DataItem* includes all DataItems for the dedicated PDU.		

4.7.4.2.7 Truth tables

Table 233 and Table 234 contains the valid combinations, their meaning, and their description of TxOption.

Table 233 – Truth table used by the PPM for TxOption for non-streams

RT_CLASS_X	ARProperties. StartupMode	IRT	MRPD	DFP	Description
		FrameSend-Offset, Port	FrameID, FrameSend-Offset, Port	SFPosition	
RT_CLASS_1, RT_CLASS_2, RT_CLASS_UDP	—	—	—	—	PPM shall create one frame
RT_CLASS_3	Legacy	Values given	—	—	PPM shall create two frames. During startup (see Annex B) one frame with the given values from the connect service and one frame with RR=128 only send in the GREEN PERIOD. After ReadyForRTC3 the additional (GREEN) frame shall be deleted.
RT_CLASS_3	Legacy	Values given	Values given	—	Not supported
RT_CLASS_3	Legacy	Values given	—	Values given	Not supported
RT_CLASS_3	Legacy	Values given	Values given	Values given	Not supported
RT_CLASS_3	Advanced	Values given	—	—	PPM shall create one frame (see Annex A)
RT_CLASS_3	Advanced	Values given	Values given	—	PPM shall create two frames (see Annex A and Annex O)
RT_CLASS_3	Advanced	Values given	—	Values given	PPM shall create one frame (see Annex A and Annex M)
RT_CLASS_3	Advanced	Values given	Values given	Values given	PPM shall create two frames (see Annex A and Annex O and Annex M)

Table 234 – Truth table used by the PPM for TxOption for streams

RT_CLASS_X	Traffic class	ARProperties. TimeAwareSystem	MRPD	Description
			VLAN, Port	
RT_CLASS_STREAM	RT	NonTimeAware	—	PPM shall create one provider for one frame (see Annex A)
RT_CLASS_STREAM	HIGH, LOW	NonTimeAware	—	Not supported
RT_CLASS_STREAM	RT	TimeAware	—	Not supported
RT_CLASS_STREAM	HIGH, LOW	TimeAware	—	PPM shall create one provider for one frame (see Annex A)
RT_CLASS_STREAM	HIGH, LOW	TimeAware	Values given	PPM shall create one provider for two frames (see Annex A and Annex O)

4.7.4.3 Consumer protocol machine

4.7.4.3.1 General

Figure 84 shows the basic structure of the Consumer Protocol Machine (CPM).

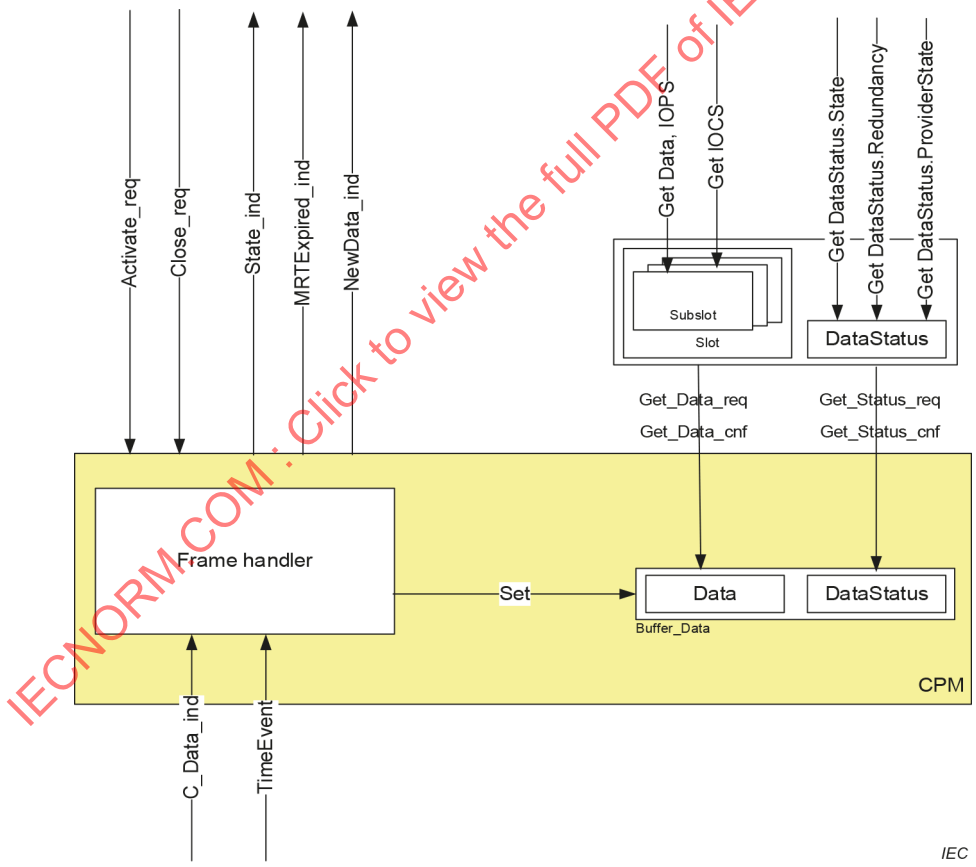


Figure 84 – Basic structure of a CPM

4.7.4.3.2 Primitive definitions

4.7.4.3.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by CPM are described in the service definition and shown in Table 235.

Table 235 – Remote primitives issued or received by CPM

Primitive	Source	Destination	Associated parameters	Functions
LMPM_C_Data.ind	LMPM	CPM	CREP, Port, DA, SA, Prio, C_SDU, APDU_Status	—
UDP_C_Data.ind	IP	CPM	CREP, IPPort, DA, SA, Prio, C_SDU, APDU_Status	—

4.7.4.3.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CPM are described in the service definition and shown in Table 236.

Table 236 – Local primitives issued or received by CPM

Primitive	Source	Destination	Associated parameters	Functions
CPM_Activate_cnf (-)	CPM	CMSU CMDMC CTLSU	CREP, ERRCLS, ERRCODE	—
CPM_Activate_cnf (+)	CPM	CMSU CMDMC CTLSU	CREP	—
CPM_Activate_req	CMSU CMDMC CTLSU	CPM	CREP, DA, SA, FrameID, RxOption, Exp_Length, DataHoldFactor, Default_Value, Default_Status	—
CPM_Close_cnf (+)	CPM	CMSU CMDMC CTLSU	CREP	—
CPM_Close_req	CMSU CMDMC CTLSU	CPM	CREP	—
CPM_Error_ind	CMSU CTLSU	CPM	CREP, ERRCLS, ERRCODE	Locally generated event if something strange happens
CPM_Get_Data_cnf (-)	CPM	FSPMDEV FSPMCTL	CREP, ERRCLS, ERRCODE	—
CPM_Get_Data_cnf (+)	CPM	FSPMDEV FSPMCTL	CREP, Data, New_Flag	—
CPM_Get_Data_req	FSPMDEV FSPMCTL	CPM	CREP	—
CPM_Get_Status_cnf (-)	CPM	FSPMDEV FSPMCTL	CREP, ERRCLS, ERRCODE	—

Primitive	Source	Destination	Associated parameters	Functions
CPM_Get_Status_cnf (+)	CPM	FSPMDEV FSPMCTL	CREP, Status, RecvCounter	—
CPM_Get_Status_req	FSPMDEV FSPMCTL	CPM	CREP	—
CPM_MRTExpired_ind	CPM	Diagnosis Module	info	—
CPM_NewData_ind	CPM	FSPMDEV FSPMCTL CMIO CTLIO CMDMC	AREP, CREP, APDU_Status, data {Data, NoData)	—
CPM_State_ind	CPM	CMIO CTLIO CMDMC	state {Start, Stop}	—

4.7.4.3.3 State transition diagram

The state transition diagram of the CPM is shown in Figure 85.

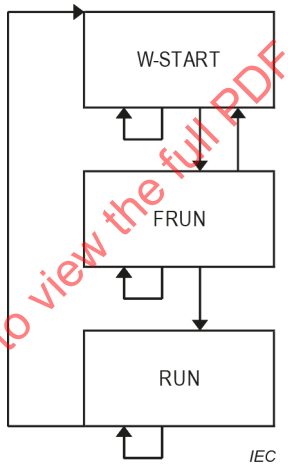


Figure 85 – State transition diagram of CPM

States of the CPM are:

W-START The W-START state indicates that the initialization is needed. The Activate service sets the machine to the RUN state or to the FRUN (first run) state.

FRUN The state machine is waiting for the first C_Data_ind service indication in these states. The DataHoldTime is ignored in this state. After passing the Start indication to the user the RUN state is entered.

RUN In this state the monitoring using the DataHoldtime is active. A timeout will force a state transition back to W-START (combined with a Stop indication).

4.7.4.3.4 State machine description

This protocol state machine is the communication endpoint for the RT_CLASS_x cyclic conveyed data. It checks the received frames against the protocol and semantics and monitors the timely reception of the expected data.

Each time a valid C_Data.ind is received, the time is read and stored in a local variable together with a flag.

Each CPM handles up to two frames with one or two FrameIDs:

- In case of NonTimeAware systems:
  - Up to two frames with the same FrameID are handled.
  - MRPD is used:
    For RT_CLASS_3 up to two frames with the same FrameID and for seamless redundancy (MRPD) two frames with two FrameIDs (FrameID and FrameID+1) are handled.
- In case of TimeAware systems:
  - Up to two frames with the same FrameID are handled.
  - MRPD is used:
    Each CPM handles up to two frames with one FrameID. For seamless redundancy (MRPD) two frames with the same FrameID, the same DestinationAddress and two different VLANs are handled.
  - End station FRER is used:
    Each CPM handles up to two frames with one FrameID. For seamless redundancy (FRER) two frames with the same FrameID, the same DestinationAddress, two different VLANs and FRER header are handled.

4.7.4.3.5 CPM state table

Table 237 contains the complete description of the CPM state table.

Table 237 – CPM state table

#	Current State	Event /Condition =>Action	Next State
1	W-START	**CPM_Activate_req ()** => Store arguments New_Data := FALSE for all ports of interface do: RxA(Port, FrameID) := Invalid DHt := 0 Cycle := 0 RecvCnt := 0 Buffer_Data := Default_Value Buffer_Status := Default_Status Primary := TRUE ControlInterval := SendClockFactor x ReductionRatio x 31,25 µs StartTimer (ControlInterval) CPM_Activate_cnf (+)	FRUN
2	W-START	**CPM_Close_req ()** => StopTimer () CPM_Close_cnf (+)	W-START
3	W-START	**CPM_Get_Status_req ()** => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Get_Status_cnf (-)	W-START
4	W-START	**CPM_Get_Data_req ()** => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Get_Data_cnf (-)	W-START

#	Current State	Event /Condition =>Action	Next State
5	W-START	**C_Data_ind** => ignore	W-START
6	FRUN	**CPM_Activate_req ()** => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Activate_cnf (-)	FRUN
7	FRUN	**CPM_Close_req ()** => StopTimer () CPM_Close_cnf (+)	W-START
8	FRUN	**CPM_Get_Status_req ()** => Status := Cycle, Buffer_Status RecvCounter := RecvCnt RecvCnt := 0 CPM_Get_Status_cnf (+)	FRUN
9	FRUN	**CPM_Get_Data_req ()** => Data := Buffer_Data New_Flag := New_Data New_Data := FALSE CPM_Get_Data_cnf (+)	FRUN
10	FRUN	**C_Data_ind** /FrameGuard () == INVALID => ignore	FRUN

11	FRUN	**C_Data_ind**
		/FrameGuard () == VALID, DHTReload, UpdateData
		=>
		New_Data := TRUE
		RxA(Port, FrameID) := DATA_CYCLE_TAG
		DHt := 0
		Cycle := DATA_CYCLE_TAG
		Buffer_Data := Filter (Data)
		Buffer_Status := DATA_STATUS_TAG
		RecvCnt := RecvCnt + 1
		CPM_State_ind (Start)
		CPM_NewData_ind (Data)
		If MRPD active then
		  StartTimer (MRWDt)
		
		//NOTE DataStatus.State == don't care and DataStatus.DataValid == Valid
			RUN

12	FRUN	**C_Data_ind**
		/FrameGuard () == VALID, DHTReload
		=>
		RxA(Port, FrameID) := DATA_CYCLE_TAG
		DHt := 0
		Cycle := DATA_CYCLE_TAG
		Buffer_Status := DATA_STATUS_TAG
		RecvCnt := RecvCnt + 1
		CPM_State_ind (Start)
		CPM_NewData_ind (NoData)
		
		If MRPD active then
		  StartTimer (MRWDt)
		
		//NOTE DataStatus.State == Backup
			RUN

13	FRUN	**TimerExpired (ControlInterval)** => ignore	FRUN
14	FRUN	**TimerExpired (MRWDt)** => ignore	FRUN

#	Current State	Event /Condition =>Action	Next State
15	RUN	**CPM_Activate_req ()** => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Activate_cnf (-)	RUN
16	RUN	**CPM_Close_req ()** => StopTimer () CPM_Close_cnf (+)	W-START
17	RUN	**CPM_Get_Status_req ()** => Status := Cycle, Buffer_Status, RecvCounter := RecvCnt, RecvCnt := 0 CPM_Get_Status_cnf (+)	RUN
18	RUN	**CPM_Get_Data_req ()** => Data := Buffer_Data, New_Flag := New_Data New_Data := FALSE CPM_Get_Data_cnf (+)	RUN
19	RUN	**C_Data_ind** /FrameGuard () == INVALID => ignore	RUN
20	RUN	**C_Data_ind/FrameGuard () == VALID, DHTReload, UpdateData** => New_Data := TRUE RxA(Port, FrameID) := DATA_CYCLE_TAG DHt := 0 Cycle := DATA_CYCLE_TAG Buffer_Data := Filter (Data) Buffer_Status := DATA_STATUS_TAG RecvCnt := RecvCnt + 1 CPM_NewData_ind (Data) //NOTE DataStatus.State == don't care and DataStatus.DataValid == Valid	RUN
21	RUN	**C_Data_ind** /FrameGuard () == VALID, DHTReload => RxA(Port, FrameID) := DATA_CYCLE_TAG DHt := 0 Cycle := DATA_CYCLE_TAG Buffer_Status := DATA_STATUS_TAG RecvCnt := RecvCnt + 1 CPM_NewData_ind (NoData) //NOTE DataStatus.State == Backup	RUN
22	RUN	**TimerExpired (ControlInterval)** /DHt > (DataHoldFactor-1) => DHt := 0 StopTimer () CPM_State_ind (Stop) //NOTE DHt expired	W-START
23	RUN	**TimerExpired (ControlInterval)** /DHt < (DataHoldFactor) => DHt := DHt +1	RUN

#	Current State	Event /Condition =>Action	Next State
24	RUN	**TimerExpired (MRWDt)** /MRPD active && MRPDGuard () == Adjunctive frame received => for all ports, FrameIDs RxA(Port, FrameID) := Invalid //NOTE No checking of the receive port intended CPM_MRTExpired_ind (Ok) StartTimer (MRWDt)	RUN
25	RUN	**TimerExpired (MRWDt)** /MRPD active && MRPDGuard () == No adjunctive frame received => for all ports, FrameIDs RxA(Port, FrameID) := Invalid //NOTE No checking of the receive port intended CPM_MRTExpired_ind (Fault) StartTimer (MRWDt)	RUN

4.7.4.3.6 Functions, Macros, Timers and Variables

Table 238 contains the functions, macros, timers and variables used by the CPM and their arguments and their descriptions.

Table 238 – Functions, Macros, Timers and Variables used by the CPM

Name	Type	Function/Meaning
Filter	Function	This local function filters from the received data the portion to be stored for the Buffer_Data
FrameGuard	Function	This local function checks the received frame against the expected structure and semantics according to the coding and the truth tables. VALID / INVALID: This flag shows whether the received frame is valid. DHTReload: This flag shows that the DataHoldTimer shall be retriggered. UpdateData: This flag shows that Buffer_Data and Buffer_Status shall be updated with the received data
MRPDGuard	Function	This local function checks whether the adjunctive frame defined by the activation of MRPD is received or not. It uses the local variable RxA as database.
StartTimer	Function	This local function starts or restarts a timer.
StopTimer	Function	This local function stops a timer.
TimerExpired	Function	This local function is issued if a timer expires.

C_Data_ind	Macro	if (RxOption == RTC)
		LMPM_C_Data.ind (CREP, Port, DA, SA, Prio, C_SDU, APDU_Status)
		
		if (RxOption == UDP)
		UDP_C_Data.ind (IPPort, CREP, DA, SA, Prio, C_SDU, APDU_Status)

DATA_CYCLE_TAG	Macro	if (!DFP)
		APDU_Status.CycleCounter
		
		if (DFP)
		SFCycleCounter

Name	Type	Function/Meaning

DATA_STATUS_TAG	Macro	if (!DFP)
		APDU_Status
		
		if (DFP)
		SFCycleCounter, DataStatus

RxOption	Macro	Contains the – RT_CLASS_x – FrameID, – if MRPD – FrameID (redundant path), – Media redundancy watchdog (MRPD active) – if DFP – subaddress SFPosition
ControlInterval	Timer	This local timer contains the monitoring interval for the DataHoldFactor. ControlInterval := SendClockFactor x ReductionRatio x 31,25 µs This timer shall use a autoreload function to start itself after expiration. It shall be aligned to the PTCP or locally driven SendClock.
MRWDt	Timer	This local timer contains the monitoring interval for the Media Redundancy WatchDog time. It is used to monitor the receiving of the adjunctive frames defined by MRPD and to issue an event if expired.
Buffer_Data	Variable	This local variable contains the last valid service data unit conveyed with a LMPM_C_Data indication.
Buffer_Status	Variable	This local variable contains the last valid APDU_Status conveyed with a LMPM_C_Data indication.
Cycle	Variable	This local variable contains the last valid Cycle Counter conveyed with a LMPM_C_Data indication with valid data.
Default_Status	Variable	The CycleCounter is set to invalid, the TransferStatus and the DataStatus is set to zero.
Default_Value	Variable	The Data is set to zero and the IOxS to BAD.
DHt	Variable	This local variable contains the DataHoldFactor counter value of the time events since last LMPM_C_Data indication.
ERRCLS	Variable	This local variable contains the entity signaling the error.
ERRCODE	Variable	This local variable contains the error reason in the context of the ERRCLS.
New_Data	Variable	This local variable signals the receipt of a valid LMPM_C_Data indication (set to FALSE by a GetData request).
RecvCnt	Variable	This local variable contains the counter value of the received cyclic PDU since last GetStatus request. This information can be used by the application as a "new data" + "age of data" hint.

RxA	Variable	This local variable contains the stored information about the received frames for the MRPD check.
		
		Principle:
		RxA(m, n) == RxA(o, n+1)
		// Stored APDU_Status.CycleCounter or SFCycleCounter are equal
		// m, o element {1..MaxPort}
		// n, n+1 element {FrameID, FrameID+1}

4.7.4.3.7 Truth tables

4.7.4.3.7.1 Truth tables for RxOption

Table 239 and Table 240 contain the valid combinations, their meaning and their description for RxOption.

Table 239 – Truth table used by the CPM for RxOption for non-streams

RT_CLASS_X	ARProperties. StartupMode	MRPD	DFP	Description
		FrameID, Redundancy watchdog	SFPosition	
RT_CLASS_1, RT_CLASS_2, RT_CLASS_UDP	—	—	—	CPM shall create one consumer for one frame
RT_CLASS_3	Legacy	—	—	CPM shall create one consumer for one frame. During startup (see Annex B) the DHT values are for RR=256 and after ReadyForRTC3 as given during the connect service.
RT_CLASS_3	Legacy	Values given	—	Not supported
RT_CLASS_3	Legacy	—	Values given	Not supported
RT_CLASS_3	Legacy	Values given	Values given	Not supported
RT_CLASS_3	Advanced	—	—	CPM shall create one consumer for one frame (see Annex A)
RT_CLASS_3	Advanced	Values given	—	CPM shall create one consumer for two frames (see Annex A and Annex O)
RT_CLASS_3	Advanced	—	Values given	CPM shall create one consumer for one frame (see Annex A)
RT_CLASS_3	Advanced	Values given	Values given	CPM shall create one consumer for two frames (see Annex A and Annex O)

Table 240 – Truth table used by the CPM for RxOption for streams

RT_CLASS_X	Traffic class	ARProperties. TimeAwareSystem	MRPD	Description
			VLAN, Redundancy watchdog	
RT_CLASS_STREAM	RT	NonTimeAware	—	CPM shall create one consumer for one frame (see Annex A)
RT_CLASS_STREAM	HIGH, LOW	NonTimeAware	—	Not supported
RT_CLASS_STREAM	RT	TimeAware	—	Not supported
RT_CLASS_STREAM	HIGH, LOW	TimeAware	—	CPM shall create one consumer for one frame (see Annex A)
RT_CLASS_STREAM	HIGH, LOW	TimeAware	Values given	CPM shall create one consumer for one frame (see Annex A and Annex O)

4.7.4.3.7.2 CPM checking rules for one frame

Table 241, Table 242, Table 243 and Table 244 contain the truth tables used by the CPM for frames.

Table 241 – Truth table for one frame using RT_CLASS_x

Input					Output
Received frame					Local
Received-InRED [c]	APDU_Status. TransferStatus	SourceMAC-Address [b]	FrameID, redundant FrameID [a]	C_SDU. Length	Frame structure
FALSE	—	—	—	—	Invalid
TRUE	Invalid	—	—	—	Invalid
TRUE	Valid	Wrong	—	—	Invalid
TRUE	Valid	Ok	Wrong	—	Invalid
TRUE	Valid	Ok	Ok	Wrong	Invalid
TRUE	Valid	Ok	Ok	Ok	Valid

[a] The SFPosition.Position (different frame layout) is managed by the DFP_RELAY.

[b] For frames using DFP optional.

[c] Only checked for RT_CLASS_3.

Table 242 – Truth table for one frame using RT_CLASS_UDP

Input					Output
Received frame					Local
APDU_Status. TransferStatus	SourceIP-Address	SourceUDP-Port	FrameID	C_SDU. Length	Frame structure
Invalid	—	—	—	—	Invalid
Valid	Wrong	—	—	—	Invalid
Valid	Ok	Wrong	—	—	Invalid
Valid	Ok	Ok	Wrong	—	Invalid
Valid	Ok	Ok	Ok	Wrong	Invalid
Valid	Ok	Ok	Ok	Ok	Valid

Table 243 – Truth table for the C_SDU

Input	Output
Received frame	Local
APDU_Status.CycleCounter	C_SDU structure
Older [a]	Invalid
Newer [b]	Valid

[a] See 4.7.2.1.2 and Table 211.
When activating a CPM, the stored CycleCounterValue is set to invalid to make the next received frame in all cases “Newer”.
The check is done against the last successfully received and stored frame or APDU_Status by this CPM.

[b] See 4.7.2.1.2 and Table 211.

Table 244 – Truth table for arranging DHt and data

Input					Output		
Received frame					Local		
Frame structure	**C_SDU structure**	**APDU_Status. DataStatus. DataValid**	**APDU_Status. DataStatus. State**	**Local Data-Transfer-Control [a]**	**Datahold timer**	**Update data**	**FrameGuard**
Invalid	—	—	—	—	—	No	Invalid
Valid	Invalid	—	—	—	—	No	Invalid
Valid	Valid	Invalid	Primary	—	—	No	Invalid
Valid	Valid	Invalid	Backup	—	Reload	No	Valid
Valid	Valid	Valid	Primary	Discard [b]	Reload	No	Valid
Valid	Valid	Valid	Backup	Discard	Reload	No	Valid
Valid	Valid	Valid	Primary	Transfer	Reload	Yes	Valid
Valid	Valid	Valid	Backup	Transfer [c]	Reload	Yes	Valid

[a] Transfer if ARType ≠ IOCARSR.

[b] Combination used during switchover.

[c] Used by the CTLSRL in combination with an IO device configured to provide valid input data independent from the AR state (GSD RT_InputOnBackupAR_Supported == TRUE).

4.7.4.3.7.3 CPM checking rules for one subframe

Table 245, Table 246, Table 247 and Table 248 contain the truth tables used by the CPM for subframes.

Table 245 – Truth table for the Subframe – frame check

Input			Output
FrameID	**SourceMAC-address**	**Header Subframe. SFCRC16**	**Frame check**
FALSE	—	—	Invalid
TRUE	FALSE	—	Invalid
TRUE	TRUE	Invalid	Invalid
TRUE	TRUE	Valid	Valid
TRUE	TRUE	Don't care	Valid

Table 246 – Truth table for the Subframe – sub frame check

Input			Output
Subframe. Position	**Subframe. Length**	**Subframe. SFCRC16**	**Sub frame check**
FALSE	—	—	Invalid
TRUE	FALSE	—	Invalid
TRUE	TRUE	Invalid	Invalid
TRUE	TRUE	Valid	Valid
TRUE	TRUE	Don't care	Valid

Table 247 – Truth table for the Subframe – sub frame data check

Input		Output
Subframe. DataStatus. Ignore	Subframe. SFCycleCounter	Sub frame data check
Ignore	—	Invalid
Ok	OLD	Invalid
Ok	NEW	Valid

Table 248 – Truth table for the Subframe – DHt and data

Input					Output		
Received frame					Local		
Frame check	Sub frame check	Sub frame data check	Subframe. DataStatus. Valid	Subframe. DataStatus. State	Datahold timer	Update data	Frame-Guard
Invalid	—	—	—	—	—	No	Invalid
Valid	Invalid	—	—	—	—	No	Invalid
Valid	Valid	Invalid	—	—	—	No	Invalid
Valid	Valid	Valid	Invalid	Backup	Reload	No	Valid
Valid	Valid	Valid	Valid	Primary	Reload	Yes	Valid
Valid	Valid	Valid	Valid	Backup	Reload	Yes	Valid

4.7.5 DLL Mapping Protocol Machines

There is no DLL Mapping Protocol Machine (DMPM) defined for this Protocol.

4.8 Real time acyclic

4.8.1 RTA syntax description

4.8.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.8.1.2 RTA APDU abstract syntax

Table 249 defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 249 shall represent the content of the DLSDU in Table 10.

Table 249 – RTA APDU syntax

APDU name	APDU structure
RTA-PDU	DATA-RTA-PDU ^ ACK-RTA-PDU ^ NACK-RTA-PDU ^ ERR-RTA-PDU ^ FREQ-RTA-PDU ^ FRSP-RTA-PDU
UDP-RTA-PDU	IPHeader, UDPHeader, FrameID, RTA-PDU
NOTE The DLPDU_Padding applies for the RTA-PDU and the UDP-RTA-PDU.	

Table 250 defines structures for substitutions of elements of the APDU structures shown in Table 249.

Table 250 – RTA substitutions

Substitution name	Structure
Reference	DestinationServiceAccessPoint, SourceServiceAccessPoint
DestinationServiceAccess-Point	AlarmEndpoint
SourceServiceAccessPoint	AlarmEndpoint
FlagsSequence	AddFlags, SendSeqNum, AckSeqNum
VendorDeviceErrorInfo	VendorID, DeviceID, Data*
DATA-RTA-DPDU	PDUType (=1), FlagsSequence, VarPartLen (1 – 1 432) [a], RTA-SDU
DATA-RTA-PDU	Reference, DATA-RTA-DPDU
ACK-RTA-APDU	PDUType (=3), FlagsSequence, VarPartLen (=0)
ACK-RTA-PDU	Reference, ACK-RTA-APDU
NACK-RTA-NPDU	PDUType (=2), FlagsSequence, VarPartLen (=0)
NACK-RTA-PDU	Reference, NACK-RTA-NPDU
ERR-RTA-EPDU	PDUType (=4), FlagsSequence, VarPartLen (=4), PNIOStatus, [VendorDeviceErrorInfo] [b]
ERR-RTA-PDU	Reference, ERR-RTA-EPDU
FREQ-RTA-FPDU	PDUType (=5), FlagsSequence, VarPartLen (4 – 1 432) [a], RSI-SDU
FREQ-RTA-PDU	Reference, FREQ-RTA-QPDU
FRSP-RTA-FPDU	PDUType (=6), FlagsSequence, VarPartLen (4 – 1 432) [a], RSI-SDU
FRSP-RTA-PDU	Reference, FRSP-RTA-FPDU
AO-RTA-PlainText	FrameID, RTA-PDU
AE-RTA-PlainText	DATA-RTA-DPDU ^ ACK-RTA-APDU ^ NACK-RTA-NPDU ^ ERR-RTA-EPDU ^ FREQ-RTA-FPDU ^ FRSP-RTA-FPDU
AE-RTA-AssociatedData	FrameID, Reference

[a] The maximum VarPartLen is calculated to fit in the RTA-PDU and in the UDP-RTA-PDU.

[b] The number of octets of this substitution shall not exceed the DLPDU_Padding. See minimum frame size in IEEE Std 802.3.

4.8.1.3 RSI APDU abstract syntax

Table 251 defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 251 represent the content of the RSI-SDU in Table 250.

Table 251 – RSI APDU syntax

APDU name	APDU structure
RSI-SDU	FOpnumOffset, FragmentOf (RSI-REQ-PDU) [a] ^ FragmentOf (RSI-RSP-PDU) [a]
RSI-REQ-PDU	RspMaxLength, RSI-CONN-PDU ^ RSI-SVCS-PDU ^ NULL [b]
RSI-RSP-PDU	PNIOStatus, [PROFINETIOServiceResPDU [c]]
[a] The macro FragmentOf() is used to create the individual fragments. [b] Only allowed with OpNum PrmWriteEnd. [c] Without IODReleaseRes.	

Table 252 defines structures for substitutions of elements of the APDU structures shown in Table 251.

Table 252 – RSI substitutions

Substitution name	Structure
RSI-CONN-PDU	VendorID, DeviceID, InstanceID, RsiInterface, Padding, (IODConnectReq [a] ^ IODReadReq [b])
RSI-SVCS-PDU	PROFINETIOServiceReqPDU [c]
[a] Shall be used with OpNum Connect. [b] Shall be used with OpNum ReadImplicit or OpNum ReadConnectionless. [c] Without IODConnectReq and IODReleaseReq.	

4.8.2 RTA transfer syntax

4.8.2.1 Coding section related to RTA-PDUs specific fields

4.8.2.1.1 Coding of the field AlarmEndpoint

This field shall be coded as data type Unsigned16 with values according to Table 253 and Table 254.

It identifies, in combination with the host address, the service access points of a bidirectional connection as shown in Figure 86. The values for source and destination shall be exchanged during a context management operation.

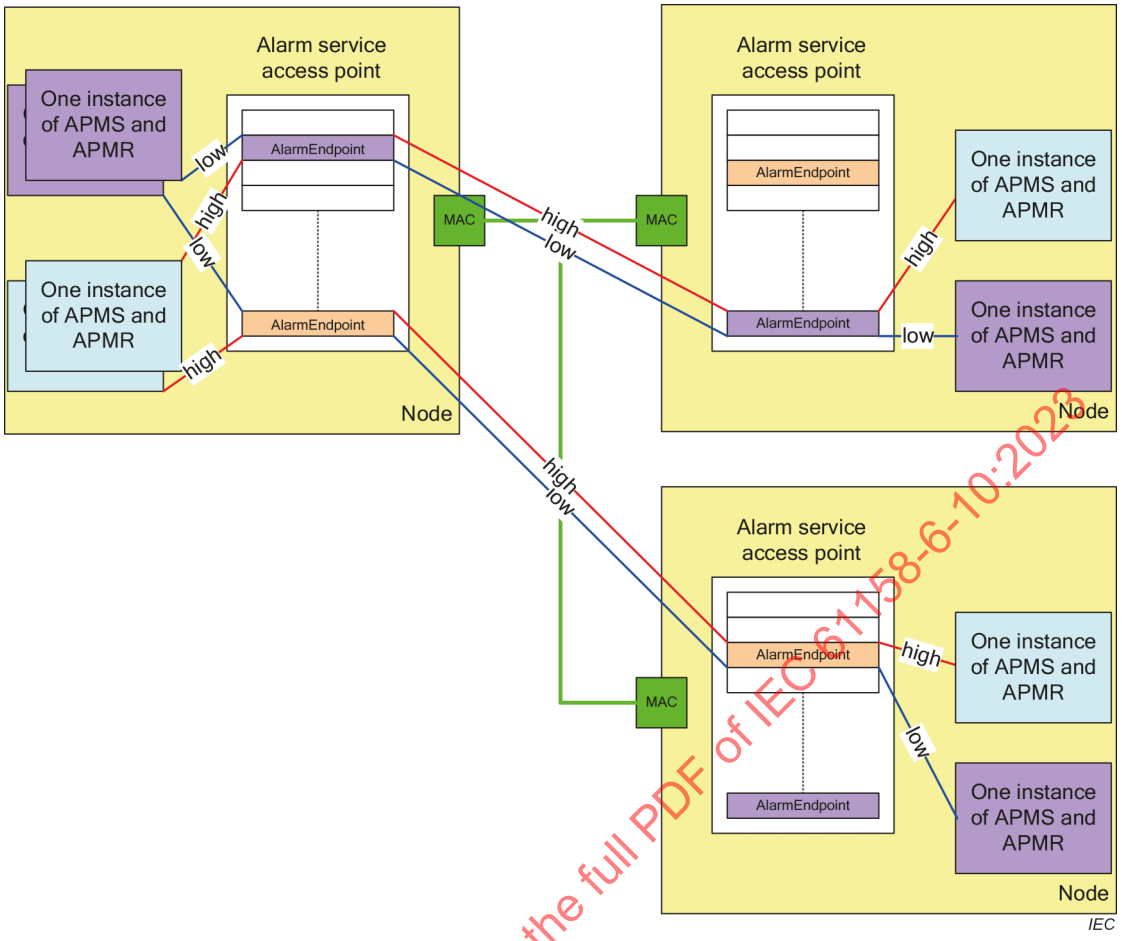


Figure 86 – Addressing scheme of RTA

NOTE The IO controller sends its AlarmEndpoint during the connect to the IO device using the field LocalAlarmReference in the AlarmCRBlockReq. The IO device responds with its AlarmEndpoint using the field LocalAlarmReference in the AlarmCRBlockRes.

The IO controller uses its AlarmEndpoint as SourceServiceAccessPoint and the AlarmEndpoint of the IO devices as DestinationServiceAccessPoint. The IO device uses its AlarmEndpoint as SourceServiceAccessPoint and the AlarmEndpoint of the IO controller as DestinationServiceAccessPoint.

The terms ISAP (Initiator ServiceAccessPoint) and RSAP (Responder ServiceAccessPoint) are used for the direction independent designation. The IO controller acts as initiator, and the IO device acts as responder.

These AlarmEndpoints are established with one of the RSI connect services and shall be used for all RSI services and for alarms.

Table 253 – AlarmEndpoint in conjunction with PDUType.Version := 1

Value (hexadecimal)	Meaning	Usage
0x0000 – 0xFFFF	SourceServiceAccessPoint or DestinationServiceAccessPoint	—

Table 254 – AlarmEndpoint in conjunction with PDUType.Version := 2

Value (hexadecimal)	Meaning	Usage
0x0000 – 0x7FFF	RSI Initiator Instance (ISAP) or RSI Responder Instance (RSAP)	These values shall be used for established RSI connections.
0x8000 – 0xFFFE	Reserved	—
0xFFFF	CON-SAP	This value shall be used for the first Request fragment of Connect, ReadImplicit or ReadConnectionless.

4.8.2.1.2 Coding of the field PDUType

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0 – 3: PDUType.Type
This field shall be coded with the values according to Table 255 and Table 256.

Table 255 – PDUType.Type with PDUType.Version := 1

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
0x01	RTA_TYPE_DATA	Shall only be used to encode the DATA-RTA-PDU
0x02	RTA_TYPE_NACK	Shall only be used to encode the NACK-RTA-PDU
0x03	RTA_TYPE_ACK	Shall only be used to encode the ACK-RTA-PDU
0x04	RTA_TYPE_ERR	Shall only be used to encode the ERR-RTA-PDU
0x05 – 0x0F	Reserved	—

Table 256 – PDUType.Type with PDUType.Version := 2

Value (hexadecimal)	Meaning	Usage
0x00 – 0x02	Reserved	—
0x03	RTA_TYPE_ACK	Shall only be used to encode the ACK-RTA-PDU. This PDU shall be used to acknowledge FREQ-RTA-PDU and FRES-RTA-PDU and for connection monitoring.
0x04	RTA_TYPE_ERR	Shall only be used to encode the ERR-RTA-PDU. RSI-Initiators and RSI-Responders send this PDU to indicate the local abort of the AR CR.
0x05	RTA_TYPE_FREQ	Shall only be used to encode the FREQ-RTA-PDU. This PDU shall be used to transfer the request from fragmented and not fragmented service.
0x06	RTA_TYPE_FRSP	Shall only be used to encode the FRSP-RTA-PDU. This PDU shall be used to transfer the response from fragmented and not fragmented service.
0x07 – 0x0F	Reserved	—

Bit 4 – 7: PDUType.Version
This field shall be coded with the values according to Table 257.

Table 257 – PDUType.Version

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Version 1 of the protocol Shall be used in conjunction with RTA and FrameID 0xFC01 or 0xFE01
0x02	Version 2 of the protocol Shall be used in conjunction with RSI and FrameID 0xFE02
0x03 – 0x0F	Reserved

4.8.2.1.3 Coding of the field AddFlags

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0 – 2: AddFlags.WindowSize
This field shall be coded with the values according to Table 258 and Table 259.

Table 258 – AddFlags.WindowSize in conjunction with PDUType.Version := 1

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Window size one
0x02 – 0x07	Reserved

Table 259 – AddFlags.WindowSize in conjunction with PDUType.Version := 2

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
0x01	Unknown WindowSize	No fragment received yet.
0x02	Smallest WindowSize	The first valid received fragment determines the receive window size of the sender.
0x03 – 0x07	Optional usable WindowSize	The first valid received fragment determines the receive window size of the sender. May be tailored by the receiver to a smaller size.

Bit 3: AddFlags.Reserved
This field shall be set according to 3.4.2.2.

Bit 4: AddFlags.TACK

This field shall be coded with the values according to Table 260 and Table 261 by the sender within a RTA-PDU to control the acknowledge behavior of the receiver.

Table 260 – AddFlags.TACK in conjunction with PDUType.Version := 1

Value (hexadecimal)	Meaning	Usage
0x00	No immediate acknowledge	Shall be set for ERR-RTA-PDU, ACK-RTA-PDU and NACK-RTA-PDU and not checked by the receiver.
0x01	Immediate acknowledge	Shall be set for immediate acknowledge of DATA-RTA-PDU.

Table 261 – AddFlags.TACK in conjunction with PDUType.Version := 2

Value (hexadecimal)	Meaning	Usage
0x00	No immediate acknowledge	Shall be used for ERR-RTA-PDU and not checked by the receiver. Shall be used for ACK-RTA-PDU, FREQ-RTA-PDU and FRES-RTA-PDU if no immediate acknowledge is required.
0x01	Immediate acknowledge	Shall be used for ACK-RTA-PDU, FREQ-RTA-PDU and FRES-RTA-PDU if immediate acknowledge is required.

Rule for generating immediate acknowledge:

unknown WindowSize Assume WindowSize 1

known WindowSize The last fragment of the window shall set immediate acknowledge, except if this fragment is the last fragment of the Request or the last fragment of the Response.

Bit 5: AddFlags.MoreFrag

This bit shall be coded with the values according to Table 262 and Table 263.

Table 262 – AddFlags.MoreFrag in conjunction with PDUType.Version := 1

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—

Table 263 – AddFlags.MoreFrag in conjunction with PDUType.Version := 2

Value (hexadecimal)	Meaning	Usage
0x00	Last fragment	Last fragment of a multi-fragmented service. Also used by services fitting in a single fragment.
0x01	More fragments follow	Used if this fragment of a multi-fragmented service is not the last fragment.

Bit 6: AddFlags.Notification

This bit shall be coded with the values according to Table 264 and Table 265.

Table 264 – AddFlags.Notification in conjunction with PDUType.Version := 1

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—

Table 265 – AddFlags.Notification in conjunction with PDUType.Version := 2

Value (hexadecimal)	Meaning	Usage
0x00	No action necessary	—
0x01	The ApplicationReadyBlock is available for reading with the service ReadNotification	ACK-RTA-PDU FRSP-RTA-PDU

Bit 7: AddFlags.Reserved
This field shall be set according to 3.4.2.2.

4.8.2.1.4 Coding of the field SendSeqNum

This field shall be coded as data type Unsigned16 with values according to Table 266 and Table 267.

Table 266 – SendSeqNum in conjunction with PDUType.Version := 1

Value (hexadecimal)	Usage
0x0000 – 0x7FFF	This value range is used for DATA-RTA-PDU after the synchronization between sender and receiver is done.
Other	Reserved
0xFFFE	This value shall be used to synchronize sender and receiver after establishment of the application relationship. Initial value which means “initial sequence, no DATA-RTA-PDU generated before”
0xFFFF	This value shall be used to synchronize sender and receiver after establishment of the application relationship. This value indicates the first DATA-RTA-PDU.

Table 267 – SendSeqNum in conjunction with PDUType.Version := 2

Value (hexadecimal)	Usage
0x0000 – 0x7FFF	This value range is used for synchronization and transmission between initiator and responder.
0x8000 – 0xFFFD	Reserved
0xFFFE	This value shall be used to synchronize initiator and responder for establishment of an application relationship. Shall be used if no FRSP-RTA-PDU or FREQ-RTA-PDU fragment has been sent before.
0xFFFF	Reserved

This field contains the number of the DATA-RTA-PDU. The incrementing and comparison of this number is done using modulo 2^{15} operations.

Table 268 and Table 269 show, as an example, the values during the initial synchronization between sender and receiver.

Table 268 – SendSeqNum and AckSeqNum start sequence in conjunction with PDUType.Version := 1

Service	Direction	Type	SendSeqNum	AckSeqNum
First AlarmNotification	IOD → IOC	DATA	0xFFFF	0xFFFE
	IOD ← IOC	ACK	0xFFFE	0xFFFF
First AlarmAck	IOD ← IOC	DATA	0xFFFF	0xFFFF
	IOD → IOC	ACK	0xFFFF	0xFFFF
Second AlarmNotification	IOD → IOC	DATA	0x0000	0xFFFF
	IOD ← IOC	ACK	0xFFFF	0x0000
Second AlarmAck	IOD ← IOC	DATA	0x0000	0x0000
	IOD → IOC	ACK	0x0000	0x0000

Table 269 – SendSeqNum and AckSeqNum start sequence in conjunction with PDUType.Version := 2

Service	Direction	Fragment	Type	SendSeqNum	AckSeqNum
ConnectReq	IOD ← IOC	1	FREQ	0x0017	0xFFFE
	IOD → IOC	–	ACK	0xFFFE	0x0017
ConnectReq	IOD ← IOC	2	FREQ	0x0018	0xFFFE
	–	–	–	–	–
ConnectRsp	IOD → IOC	1	FRSP	0x0017	0x0018
	–	–	–	–	–
WriteMultipleReq	IOD ← IOC	1	FREQ	0x0019	0x0017
	–	–	–	–	–
WriteMultipleReq	IOD ← IOC	2	FREQ	0x001A	0x0017
	IOD → IOC	–	ACK	0x0017	0x001A
WriteMultipleReq	IOD ← IOC	3	FREQ	0x001B	0x0017
	–	–	–	–	–
WriteMultipleRsp	IOD → IOC	1	FRSP	0x0018	0x001B
	–	–	–	–	–
ReadNotification	IOD → IOC	–	ACK	0x0018	0x001B
	–	–	–	–	–
ReadNotificationReq	IOD ← IOC	1	FREQ	0x001C	0x0018
	–	–	–	–	–
ReadNotificationRsp	IOD → IOC	1	FRSP	0x0019	0x001C
	–	–	–	–	–
ReadNotificationRsp	IOD → IOC	2	FRSP	0x001A	0x001C
	–	–	–	–	–

4.8.2.1.5 Coding of the field AckSeqNum

This field shall be coded as data type Unsigned16 with values according to Table 270 and Table 271.

Table 270 – AckSeqNum in conjunction with PDUType.Version := 1

Value (hexadecimal)	Usage
0x0000 – 0x7FFF	This value range is used for DATA-RTA-PDU after the synchronization between sender and transmitter is done.
Other	Reserved
0xFFFE	This value shall be used to synchronize sender and receiver after establishment of the application relationship. Initial value which means “initial sequence, no DATA-RTA-PDU received before”
0xFFFF	This value acknowledges the reception of the first DATA-RTA-PDU.

Table 271 – AckSeqNum in conjunction with PDUType.Version := 2

Value (hexadecimal)	Usage
0x0000 – 0x7FFF	This value range is used for synchronization and transmission between initiator and responder.
0x8000 – 0xFFFD	Reserved
0xFFFE	This value shall be used to synchronize initiator and responder for establishment of an application relationship. Shall be used if no FRSP-RTA-PDU or FREQ-RTA-PDU fragment has been received before.
0xFFFF	Reserved

This field contains the number of the DATA-RTA-PDU that is expected to be acknowledged or which is acknowledged.

4.8.2.1.6 Coding of the field VarPartLen

This field shall be coded as data type Unsigned16 according to Table 272. The VarPartLen shall not contain the DLPDU_Padding.

Table 272 – VarPartLen

Value (hexadecimal)	Meaning
0x0000	No RTA-SDU or RSI-SDU exists
0x0001 – 0x0598	An RTA-SDU or RSI-PDU with VarPartLen octets exists
0x0599 – 0xFFFF	Reserved

4.8.2.2 Coding section related to RSI-PDUs specific fields

4.8.2.2.1 Coding of the field FopnumOffset

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 23: FopnumOffset.Offset
This field shall be coded with the values according to Table 273.

Table 273 – FopnumOffset.Offset

Value (hexadecimal)	Meaning	Usage
0x00000000	First fragment	First fragment of a multi-fragmented service. Also used by services fitting in a single fragment.
0x00000001 – 0x00000003	Reserved	Reserved
0x00000004 – 0x00FFFFFF	Not first fragment	The offset of this fragment of a multi-fragmented service. The recipient shall only accept fragments with FopnumOffset.Offset == next-offset.
NOTE 1 The offset "next-offset" is calculated after the fragment has been received. next-offset := PDU.FopnumOffset.Offset + PDU.VarPartLen – 4. NOTE 2 4 is equal to the size of FopnumOffset. NOTE 3 The size of the PROFINETIOServiceReqPDU (or the PROFINETIOServiceResPDU) is equal to size-of-PDU := PDU.FopnumOffset.Offset + PDU.VarPartLen – 4 – RsiHeaderSize.		

Bit 24 – 28: FopnumOffset.OpNum
This field shall be coded with the values according to Table 274.

Table 274 – FopnumOffset.OpNum

Value (hexadecimal)	Meaning	equates to RPCOperationNmb	Usage
0x00	Connect	Connect	RSI-CONN-PDU
0x01	Reserved	Release [d]	—
0x02	Read	Read	RSI-SVCS-PDU only valid with ARUUID <> 0
0x03	Write	Write	RSI-SVCS-PDU
0x04	Control	Control	RSI-SVCS-PDU
0x05	ReadImplicit	ReadImplicit	RSI-CONN-PDU only valid with ARUUID == 0
0x06	ReadConnectionless	—	RSI-CONN-PDU only valid with ARUUID <> 0
0x07	ReadNotification [a]	—	RSI-SVCS-PDU
0x08	PrmWriteMore [b c]	—	RSI-SVCS-PDU
0x09	PrmWriteEnd [b c]	—	RSI-SVCS-PDU ^ NULL
0x0A	SecurityAssociationControl	SecurityAssociationControl	Reserved for security IOXSecureReq ^ IOXSecureRsp
0x0B – 0x1F	Reserved	—	—

[a] ReadNotification is a special Read service to read the ApplicationReadyBlock.

[b] PrmWriteMore and PrmWriteEnd shall be used instead of Write within a Prm phase. This means that the service Write is dedicated to the FSPM.

[c] PrmWriteEnd shall be used instead of PrmWriteMore to transfer the last record of a Prm phase. This means that an explicit service PrmEnd is not necessary and shall not be transfered from the IO controller.

[d] The service Release is replaced with an ERR-RTA-PDU (with PNIOStatus "Release").

Bit 29 – 31: FopnumOffset.CallSequence
Each fragment of an RSI call (Request or Response) has the same unique sequence number. This field shall be coded with the values according to Table 275.

Table 275 – FopnumOffset.CallSequence

Value (hexadecimal)	Meaning
0x0 – 0x7	Allowed values

NOTE The controller should start with zero and increment by 1 with every RSI call. The device shall take the CallSequence from the first fragment of the RSI call. The device shall not check this CallSequence against the previous RSI call.

4.8.2.2.2 Coding of the field RspMaxLength

This field shall be coded as data type Unsigned32 with values according to Table 276. The value shall be set to the maximum buffer size available for the response.

Table 276 – RspMaxLength

Value (hexadecimal)	Meaning
0x00000004 – 0x00FFFFFF	Usable
others	Reserved

4.8.2.2.3 Coding of the field RsiInterface

This field shall be coded as data type Unsigned8. The values shall be set according to Table 277.

Table 277 – RsiInterface

Value (hexadecimal)	Meaning	Usage	equates to RPCInterfaceUUID
0x00	IO device interface	AR or ReadConnectionless to the IO device	UUID_IO_DeviceInterface
0x01	Read Implicit IO device interface	ReadImplicit to the IO device	UUID_IO_DeviceInterface with OpNum ReadImplicit
0x02	CIM interface	AR or ReadConnectionless to the CIM	UUID_IO_CIMInterface
0x03	Read Implicit CIM interface	ReadImplicit to the CIM	UUID_IO_CIMInterface with OpNum ReadImplicit
0x04 – 0xFF	Reserved	Reserved	–

The relationship between OpNum and RsiInterface is described in Table 278.

Table 278 – Relationship between OpNum and RsiInterface

OpNum	IO device interface	ReadImplicit IO device interface	CIM interface	ReadImplicit CIM interface
Connect	CheckVDI [a]	—	CheckVDI	—
Reserved	—	—	—	—
Read	useable	—	useable	—
Write	useable	—	useable	—
Control	useable	—	useable	—
ReadImplicit	—	MatchVDI [b]	—	MatchVDI
ReadConnectionless	CheckVDI	—	CheckVDI	—
ReadNotification	useable	—	—	—
PrmWriteMore	useable	—	—	—
PrmWriteEnd	useable	—	—	—
SecurityAssociationControl	CheckVDI	—	—	—

[a] CheckVDI means that RsiInferface, VendorID, DeviceID and InstanceID of the received RSI-CONN-PDU shall be identical to the locally instantiated responder instance (see RSI_R_Add).

[b] MatchVDI means that the VendorID, DeviceID and InstanceID of the received RSI CONN PDU should match to the associated RsiInterface instance. The implementation may use a rule other than equality to look up the associated RsiInterface instance.

4.8.3 FAL Service Protocol Machines

There is no FAL Service Protocol Machine (FSPM) defined for this Protocol.

4.8.4 Application Relationship Protocol Machines

4.8.4.1 General

The symmetrical RTA transport is used by the symmetrical Acyclic Protocol Machine (APM). The APM is built as shown in Figure 87. An ALPMI / ALPMR channel is addressed by Source-Node, Destination-Node, Source-SAP, Destination-SAP and FrameID high or low.

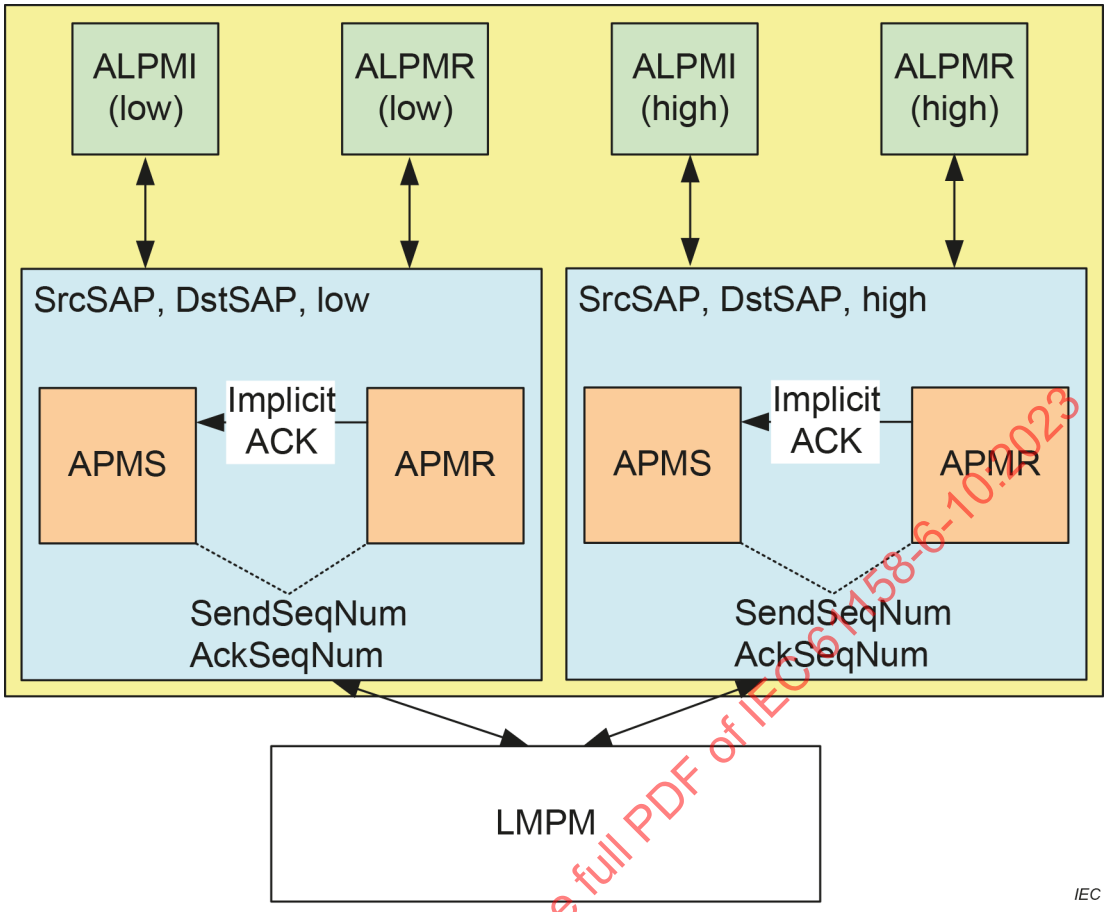


Figure 87 – Structure of the APM

The asymmetrical RSI transport is used by the Remote Service Interface Initiator (RSII) and the Remote Service Interface Responder (RSIR). The RSI is built as shown in Figure 88. An RSII / RSIR channel is addressed by DestinationAddress, SourceAddress, FrameID RSI, DestinationServiceAccessPoint and SourceServiceAccessPoint.

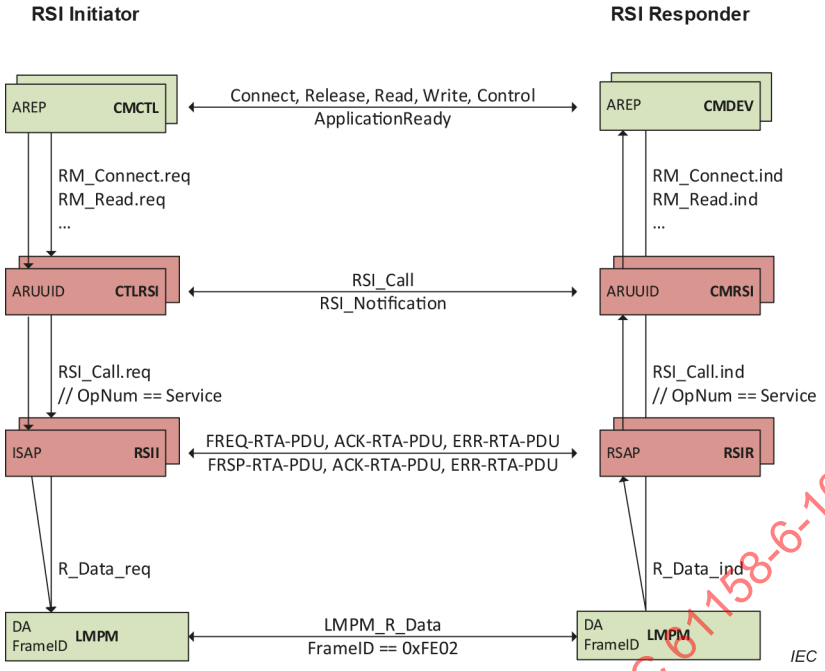


Figure 88 – Structure of the RSI

4.8.4.2 Acyclic Protocol Machine Sender

4.8.4.2.1 General

Figure 89 shows the structure of the Acyclic Protocol Machine Sender (APMS).

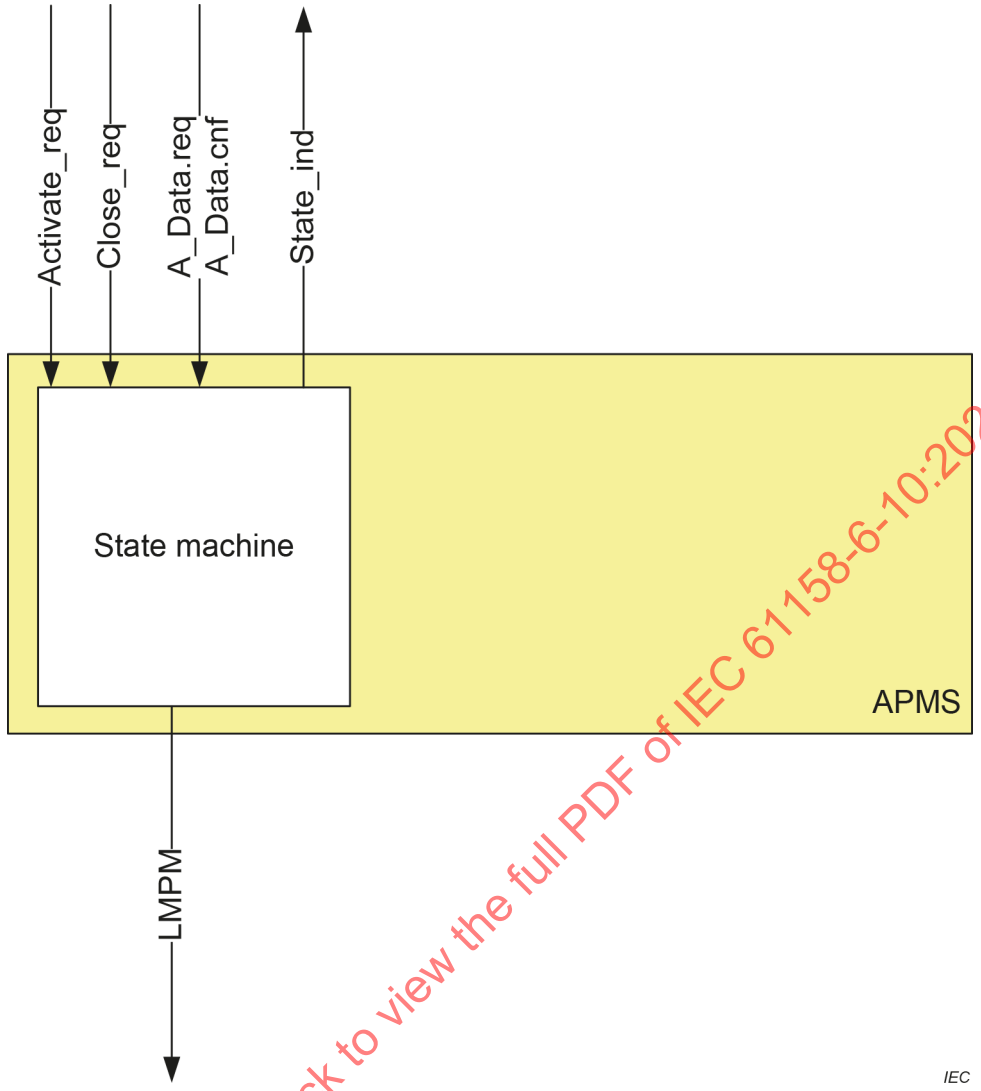


Figure 89 – Structure of the APMS

4.8.4.2.2 Primitive definitions

4.8.4.2.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by APMS are described in the service definition and shown in Table 279.

Table 279 – Remote primitives issued or received by APMS

Primitive	Source	Destination	Associated parameters	Functions
APMS_A_Data.cnf (-)	APMS	ALPMI ALPMR	CREP, ERRCLS, ERRCODE	—
APMS_A_Data.cnf (+)	APMS	ALPMI ALPMR	CREP	—
APMS_A_Data.req	ALPMI ALPMR	APMS	CREP, Data	—
LMPM_A_Data.cnf	LMPM	APMS	CREP, LMPM_status	
LMPM_A_Data.ind	LMPM	APMS	CREP, S_Port, Tstamp, DA, SA, VLANPrio, VLANID, A_SDU	—
LMPM_N_Data.cnf	LMPM	APMS	CREP, LMPM_status	—
LMPM_N_Data.ind	LMPM	APMS	CREP, DA, SA, VLANPrio, VLANId, N_SDU	—
LMPM_A_Data.req	APMS	LMPM	CREP, S_Port, Tstamp, DA, SA, VLANPrio, VLANID, A_SDU	—
LMPM_N_Data.req	APMS	LMPM	CREP, DA, SA, VLANPrio, VLANId, N_SDU	—

4.8.4.2.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by APMS are described in the service definition and shown in Table 280.

Table 280 – Local primitives issued or received by APMS

Primitive	Source	Destination	Associated parameters	Functions
APMS_Activate_req	ALPMI ALPMR	APMS	CREP, DA, SA, FrameID, VLANPrio, VLANID, RTATimeoutFactor, RTARetries, Transport, DstIP, SrcIP	-
APMS_Close_req	ALPMI ALPMR	APMS	CREP, ErrCode	—
APMS_Close_cnf	APMS	ALPMI ALPMR	CREP	—
APMS_Activate_cnf (-)	APMS	ALPMI ALPMR	CREP, ERRCLS, ERRCODE	—
APMS_Activate_cnf (+)	APMS	ALPMI ALPMR	CREP	—
APMS_Error_ind	APMS	CMSU CTLSU	CREP, ERRCLS, ERRCODE	—

4.8.4.2.3 State transition diagram

The state transition diagram of the APMS is shown in Figure 90.

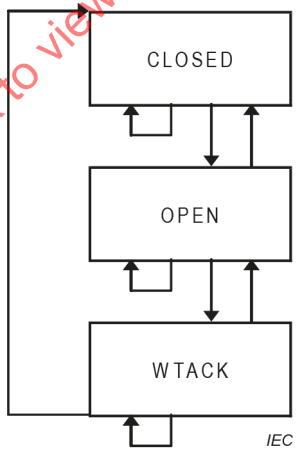


Figure 90 – State transition diagram of APMS

States of the APMS are:

CLOSED The CLOSED state indicates that an initialization is needed. Activate service sets the machine to the OPEN state.

OPEN Waiting for A-Data service request.

WTACK After issuing the transmission of data the WTACK state is entered to wait for a transport acknowledge. Receiving the acknowledge will return the machine to the OPEN state.

4.8.4.2.4 State machine description

This state machine, addressed by Source-Host, Source-ServiceAccessPoint, Destination-Host, DestinationAccessPoint and Priority, handles the A-Data service. Each priority (High and Low) is handled independently. This state machine is part of the APMS and APMR pair which is used as sender and responder for the transport acknowledge A-Data services. The A-Data services are used for the Alarm transport.

A LMPM_A_Data.ind primitive will be accepted if the value of the PDUType.Type field is DATA-RTA-PDU, ACK-RTA-PDU or NACK-RTA-PDU and the PDUType.Version is RTA_VERS=1.

The APMS state machine specifies the abort sequence and shows the corresponding ERR-RTA-PDU. These PDUs shall be sent by the APMS only for low priority alarms from both sides of the connection.

A NAK does not count as a TACK and thus a retransmission takes place. If the retransmission is again acknowledged by NAKs, a timeout for TACK will happen and an error shall be issued.

4.8.4.2.5 APMS state table

Table 281 contains the description of the APMS state machine.

Table 281 – APMS state table

#	Current State	Event /Condition =>Action	Next State
1	CLOSED	**APMS_Activate_req ()** => AlarmInstance[SrcSAP, DstSAP, Prio] := DA, RTATimeoutFactor, RTARetries, ... // NOTE Information used for ALPMI, ALPMR, APMS and APMR Send_Seq_Count:=0xFFFF Send_Seq_CountO:=0xFFFE Use CREP to identify the AlarmInstance APMS_Activate_cnf (+)	OPEN
2	CLOSED	**APMS_Close_req ()** => APMS_Close_cnf ()	CLOSED
3	CLOSED	**APMS_A_Data.req ()** => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_A_Data.cnf (-)	CLOSED
4	CLOSED	**A_Data_ind** => ignore	CLOSED
5	CLOSED	**A_Data_cnf** => ignore	CLOSED
6	OPEN	**APMS_Activate_req ()** => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_Activate_cnf (-)	OPEN

#	Current State	Event /Condition =>Action	Next State
7	OPEN	**APMS_Close_req ()** /VLANPrio == Low && ! ARusesRSI () => PDU.Version:=1 PDU.PDUType:=ERR PDU.AddFlags.TACK:=0 PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Send_Seq_Count PDU.AckSeqNum:=Expected_Seq_CountO (from APMR) PDU.PNIOStatus.ErrorCode = 0xCF PDU.PNIOStatus.ErrorDecode = 0x81 PDU.PNIOStatus.ErrorCode1 = 0xFD PDU.PNIOStatus.ErrorCode2 = ErrCode from APMS_Close request A_Data_req StopTimer (RTATimeout) APMS_Close_cnf ()	CLOSED
8	OPEN	**APMS_Close_req ()** /VLANPrio == High ‖ (VLANPrio == Low && ARusesRSI ()) => //NOTE Error PDU only for the APMS instance “low” StopTimer (RTATimeout) APMS_Close_cnf ()	CLOSED
9	OPEN	**APMS_A_Data.req ()** => PDU.Version:=1 PDU.PDUType:=DATA PDU.AddFlags.TACK:=Tack PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Send_Seq_Count PDU.AckSeqNum:=Expected_Seq_CountO (from APMR) PDU.RTA-SDU:=Data Store PDU.Flags, PDU.RTA-SDU Retry:= RTARetries A_Data_req	WTACK
10	OPEN	**A_Data_ind** /PDU.PDUType == DATA ‖ PDU.PDUType == NAK ‖ PDU.PDUType == ACK => ignore	OPEN
11	OPEN	**A_Data_ind** /PDU.Type == ERR => ERRCLS := PDU.PNIOStatus.ErrorCode1 ERRCODE := PDU.PNIOStatus.ErrorCode2 APMS_Error_ind ()	OPEN
12	OPEN	**A_Data_cnf** => ignore	OPEN
13	OPEN	**TimerExpired** => ignore	OPEN
14	WTACK	**APMS_Activate_req ()** => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_Activate_cnf (-)	WTACK

#	Current State	Event /Condition =>Action	Next State
15	WTACK	**APMS_Close_req ()** /VLANPrio == Low && ! ARusesRSI () => PDU.Version:=1 PDU.PDUType:=ERR PDU.AddFlags:=0 PDU.Window:=1 PDU.SendSeqNum:=Send_Seq_Count PDU.AckSeqNum:=Expected_Seq_CountO (from APMR) PDU.PNIOStatus.ErrorCode = 0xCF PDU.PNIOStatus.ErrorDecode = 0x81 PDU.PNIOStatus.ErrorCode1 = 0xFD PDU.PNIOStatus.ErrorCode2 = ErrCode from APMS_Close request A_Data_req StopTimer (RTATimeout) APMS_Close_cnf ()	CLOSED
16	WTACK	**APMS_Close_req ()** /VLANPrio == High ∥ (VLANPrio == Low && ARusesRSI ()) => StopTimer (RTATimeout) APMS_Close_cnf ()	CLOSED
17	WTACK	**APMS_A_Data.req ()** => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_A_Data.cnf (-)	WTACK
18	WTACK	**A_Data_ind** /(PDU.PDUType == DATA ∥ PDU.PDUType == ACK) && PDU.AckSeqNum == Send_Seq_Count => Send_Seq_CountO:=Send_Seq_Count Send_Seq_Count:= (Send_Seq_Count+1) & 0x7FFF //NOTE TACK shall be done by ACK or DATA APMS_A_Data.cnf (+)	OPEN
19	WTACK	**A_Data_ind** /(PDU.PDUType == DATA ∥ PDU.PDUType == ACK) && PDU.AckSeqNum != Send_Seq_Count => ignore	WTACK
20	WTACK	**A_Data_ind** /PDU.Type == ERR => APMS_Error_ind ()	WTACK
21	WTACK	**A_Data_ind** /PDU.PDUType == NAK => ignore //NOTE A NAK does not count as a TACK and thus a retransmission takes place	WTACK
22	WTACK	**A_Data_cnf** => StartTimer (RTATimeout)	WTACK
23	WTACK	**TimerExpired** / Retry != 0 => PDU.Version:=1 PDU.PDUType:=DATA PDU.AddFlags.TACK:=from Stored PDU.AddFlags PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Send_Seq_Count PDU.AckSeqNum:=Expected_Seq_CountO (from APMR) PDU.RTA-SDU:=from Stored PDU.RTA-SDU Retry:= Retry-1 A_Data_req	WTACK

#	Current State	Event /Condition =>Action	Next State
24	WTACK	**TimerExpired** /Retry == 0 => ERRCLS := APMS ERRCODE := TIMEOUT APMS_Error_ind () APMS_A_Data.cnf (-)	OPEN

4.8.4.2.6 Functions, Macros, Timers and Variables

Table 282 contains the functions, macros, timers and variables used by the APMS, their arguments and their descriptions.

Table 282 – Functions, Macros, Timers and Variables used by the APMS

Name	Type	Function/Meaning
StartTimer	Function	This function is used to start or restart a timer.
StopTimer	Function	This function is used to stop a timer.
TimerExpired	Function	This function signals that a timer has expired.

A_Data_cnf	Macro	if (Transport == RTA)
		LMPM_A_Data.cnf (CREP, D_Port, Tstamp, LMPM_status)
		
		if (Transport == UDP)
		LMPM_N_Data.cnf (CREP, D_Port, Tstamp, LMPM_status)

A_Data_ind	Macro	if (Transport == RTA)
		LMPM_A_Data.ind (CREP, S_Port, Tstamp, DA, SA, VLANPrio, VLANID, A_SDU)
		PDU := A_SDU
		
		if (Transport == UDP)
		LMPM_N_Data.ind (CREP, DA, SA, VLANPrio, VLANId, N_SDU)
		PDU := N_SDU

A_Data_req	Macro	if (Transport == RTA)
		A_SDU := PDU
		LMPM_A_Data.req (CREP, D_Port := AUTO, DA, SA, VLANPrio, VLANID, A_SDU)
		if (Transport == UDP)
		N_SDU := PDU
		LMPM_N_Data.req (CREP, D_Port := AUTO, DA, SA, VLANPrio, VLANID, N_SDU)

RTATimeout	Timer	This timer monitors the transport acknowledge from the responder
ERRCLS	Variable	This local variable contains the entity signaling the error.
ERRCODE	Variable	This local variable contains the error reason in the context of the ERRCLS.
Expected_Seq_CountO	Variable	This local variable contains the previous counter value of the Seq_Count variable.
RTARetries	Variable	This local variable contains the maximum retry factor from RTA ASE. The default value is 3.
Retry	Variable	This local variable is used as a working counter for the retry.
RTATimeoutFactor	Variable	This local variable contains the value from the RTA ASE. The default value is 100 ms.
Send_Seq_Count	Variable	This local variable contains the counter value which shall be used at the conveyance of the next DATA-RTA-PDU.
ARusesRSI	Macro	True if CMInitiatorObjectUUID starts with "DEA00001"

4.8.4.3 Acyclic Protocol Machine Receiver

4.8.4.3.1 General

In the following the structure of the Acyclic Protocol Machine Receiver (APMR) is shown (see Figure 91).

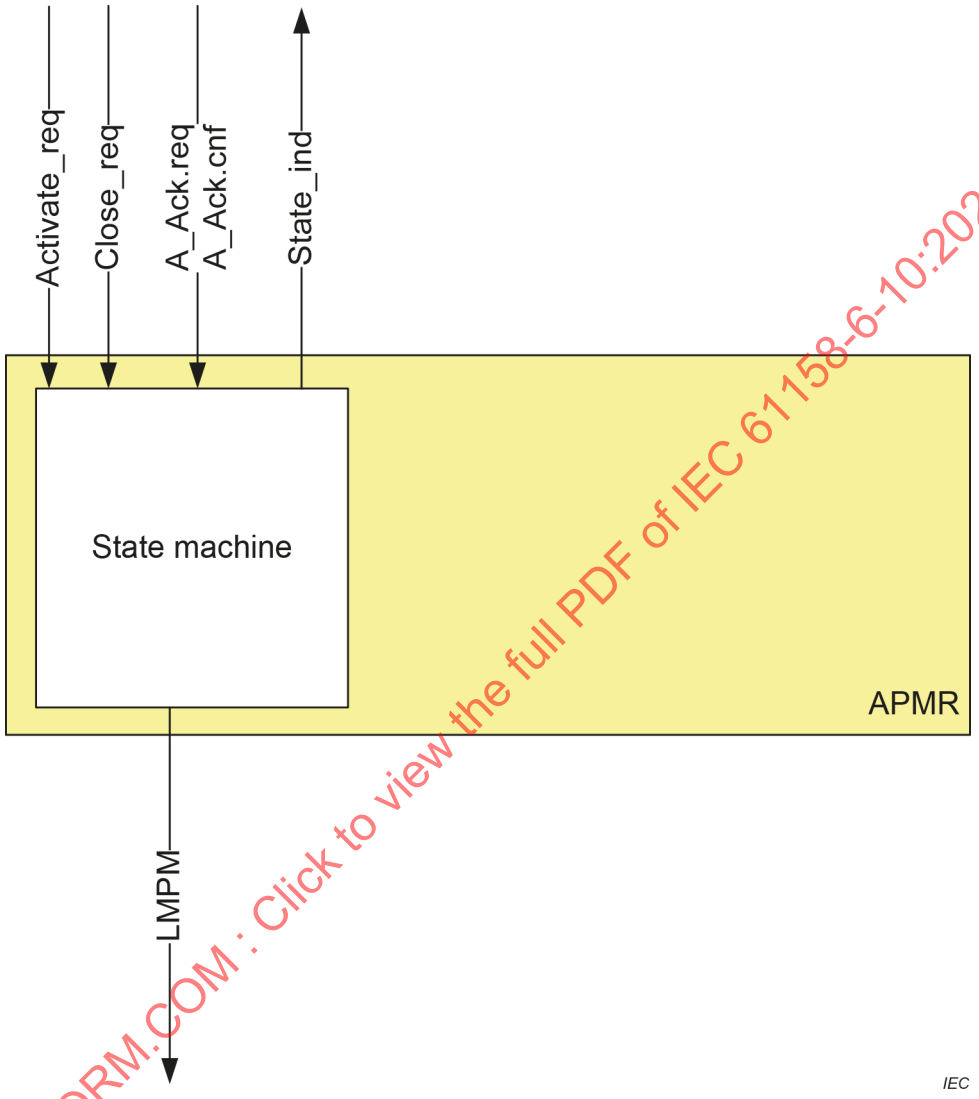


Figure 91 – Structure of the APMR

4.8.4.3.2 Primitive definitions

4.8.4.3.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by APMR are described in the service definition and shown in Table 283.

Table 283 – Remote primitives issued or received by APMR

Primitive	Source	Destination	Associated parameters	Functions
APMR_A_Data.ind	APMR	ALPMR	CREP, Data	Destination used if Data == AlarmNotification-PDU || Data == unknown
APMR_A_Data.ind	APMR	ALPMI	CREP, Data	Destination used if Data == AlarmAck-PDU || Data == unknown
LMPM_A_Data.cnf	APMR	LMPM	CREP, D_Port, Tstamp, LMPM_status	—
LMPM_A_Data.ind	APMR	LMPM	CREP, S_Port, Tstamp, DA, SA, VLANPrio, VLANID, A_SDU	—
LMPM_A_Data.req	APMR	LMPM	CREP, D_Port:=AUTO, DA, SA, VLANPrio, VLANID, A_SDU	—
LMPM_N_Data.cnf	APMR	LMPM	CREP, D_Port, Tstamp, LMPM_status	—
LMPM_N_Data.ind	APMR	LMPM	CREP, DA, SA, VLANPrio, VLANId, N_SDU	—
LMPM_N_Data.req	APMR	LMPM	CREP, D_Port:=AUTO, DA, SA, VLANPrio, VLANID, N_SDU	—

4.8.4.3.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by APMR are described in the service definition and shown in Table 284.

Table 284 – Local primitives issued or received by APMR

Primitive	Source	Destination	Associated parameters	Functions
APMR_Activate_req	CTLSU CMSU	APMR	CREP, DA, SA, FrameID, VLANPrio, VLANID, Transport, DstIP, SrcIP	—
APMR_Close_req	CTLSU CMSU	APMR	CREP	—

Primitive	Source	Destination	Associated parameters	Functions
APMR_Activate_cnf (-)	APMR	CTLSU CMSU	CREP, ERRCLS, ERRCODE	—
APMR_Activate_cnf (+)	APMR	CTLSU CMSU	CREP	—
APMR_Close_cnf (+)	APMR	CTLSU CMSU	CREP	—
APMR_Error_ind	APMR	CMSU CTLSU	CREP, ERRCLS, ERRCODE	—

4.8.4.3.3 State transition diagram

The state transition diagram of the APMR is shown in Figure 92.

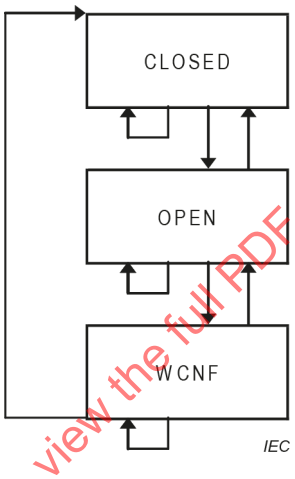


Figure 92 – State transition diagram of APMR

States of the APMR are:

CLOSED The CLOSED state indicates that an initialization is needed. The Activate service sets the machine to the OPEN state.

OPEN Waiting for LMPM_A_Data or a UDP_A_Data service indication and convey a transport acknowledge.

WCNF Wait for the confirmation of the transport acknowledge and issue a APMR_Data indication.

4.8.4.3.4 State machine description

This state machine, addressed by Source-Host, Source-ServiceAccessPoint, Destination-Host, DestinationAccessPoint and Priority, handles the A-Data service. Each priority (High and Low) is handled independently. This state machine is part of the APMS and APMR pair which is used as sender and responder for the transport acknowledge A-Data services. The A-Data service are used for the Alarm transport.

A LMPM_A_Data.ind primitive will be accepted:

– If PDUType.Type field is DATA-RTA-PDU and the PDUType.Version field is one and the VarPartLen is greater than zero and the AddFlagsWindow is one.

– If PDUType.Type field is ERR-RTA-PDU and the PDUType.Version field is one and the VarPartLen is 4.

4.8.4.3.5 APMR state table

Table 285 contains the description of the APMR state machine.

Table 285 – APMR state table

#	Current State	Event /Condition =>Action	Next State
1	CLOSED	**APMR_Activate_req ()** => AlarmInstance[SrcSAP, DstSAP, Prio] := DA, RTATimeoutFactor, RTARetries, ... //NOTE Information used for ALPMI, ALPMR, APMS and APMR Expected_Seq_Count:=0xFFFF Expected_Seq_CountO:=0xFFFE Use CREP to identify the AlarmInstance APMR_Activate_cnf (+)	OPEN
2	CLOSED	**APMR_Close_req ()** => APMR_Close_cnf (+)	CLOSED
3	CLOSED	**A_Data_ind** => ignore	CLOSED
4	CLOSED	**A_Data_cnf** => ignore	CLOSED
5	OPEN	**APMR_Activate_req ()** => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_Activate_cnf (-)	OPEN
6	OPEN	**APMR_Close_req ()** => APMR_Close_cnf (+)	CLOSED
7	OPEN	**A_Data_ind** /PDU.Type == DATA && PDU.AddFlags.Tack && PDU.SendSeqNum == Expected_Seq_Count => Store Data := PDU.RTA-SDU Expected_Seq_CountO :=Expected_Seq_Count Expected_Seq_Count := (Expected_Seq_Count+1) & 0x7FFF PDU.Version :=1 PDU.Type := ACK PDU.AddFlags.TACK :=0 PDU.AddFlags.WindowSize :=1 PDU.SendSeqNum := Send_Seq_CountO (from the APMS) PDU.AckSeqNum := Expected_Seq_CountO //NOTE The PDU.SendSeqNum shall be set to the value from the corresponding APMS A_Data_req	WCNF
8	OPEN	**A_Data_ind** /PDU.Type == DATA && PDU.AddFlags.Tack && PDU.SendSeqNum == Expected_Seq_CountO => PDU.Version:=1 PDU.Type:= ACK PDU.AddFlags.TACK:=0 PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Send_Seq_CountO (from the APMS) PDU.AckSeqNum:=Expected_Seq_CountO A_Data_req	OPEN

#	Current State	Event /Condition =>Action	Next State
9	OPEN	**A_Data_ind** /PDU.Type == DATA && PDU.AddFlags.Tack && PDU.SendSeqNum != Expected_Seq_Count && PDU.SendSeqNum != Expected_Seq_CountO => ERRCLS := RTA_ERR_CLS_PROTOCOL ERRCODE := RTA_ERR_CODE_SEQ CreateLogBookEntry () PDU.Version:=1 PDU.Type:= NAK PDU.AddFlags.TACK:=0 PDU.AddFlags.Window:=1 PDU.SendSeqNum:=Send_Seq_CountO (from the APMS) PDU.AckSeqNum:=Expected_Seq_CountO A_Data_req //NOTE A NAK does not count as a TACK and thus a retransmission takes place	OPEN
10	OPEN	**A_Data_ind** /PDU.Type == ERR => ERRCLS := PDU.PNIOStatus.ErrorCode1 ERRCODE := PDU.PNIOStatus.ErrorCode2 APMR_Error_ind ()	OPEN
11	OPEN	**A_Data_ind** /(PDU.Type == DATA && !PDU.AddFlags.Tack) => ignore	OPEN
12	OPEN	**A_Data_ind** /PDU.Type == ACK || PDU.Type == NAK => ignore	OPEN
13	OPEN	**A_Data_cnf** => ignore	OPEN
14	WCNF	**APMR_Activate_req ()** => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_Activate_cnf (-)	WCNF
15	WCNF	**APMR_Close_req ()** => APMR_Close_cnf (+)	CLOSED
16	WCNF	**A_Data_cnf** => Retrieve Data := PDU.RTA-SDU APMR_A_Data.ind ()	OPEN
17	WCNF	**A_Data_ind** /PDU.Type != ERR => ignore	WCNF
18	WCNF	**A_Data_ind** /PDU.Type == ERR => APMR_Error_ind ()	WCNF

4.8.4.3.6 Functions, Macros, Timers and Variables

Table 286 contains the functions, macros, timers and variables used by the APMR and their arguments and their descriptions.

Table 286 – Functions, Macros, Timers and Variables used by the APMR

Name	Type	Meaning

A_Data_ind	Macro	if (Transport == RTA)
		LMPM_A_Data.ind (CREP, S_Port, Tstamp, DA, SA, VLANPrio, VLANID, A_SDU)
		PDU := A_SDU
		
		if (Transport == UDP)
		LMPM_N_Data.ind (CREP, DA, SA, VLANPrio, VLANId, N_SDU)
		PDU := N_SDU

A_Data_cnf	Macro	if (Transport == RTA)
		LMPM_A_Data.cnf (CREP, D_Port, Tstamp,LMPM_status)
		
		if (Transport == UDP)
		LMPM_N_Data.cnf (CREP, D_Port, Tstamp, LMPM_status)

A_Data_req	Macro	if (Transport == RTA)
		A_SDU := PDU
		LMPM_A_Data.req (CREP, D_Port := AUTO, DA, SA, VLANPrio, VLANID, A_SDU)
		
		if (Transport == UDP)
		N_SDU := PDU
		LMPM_N_Data.req (CREP, D_Port := AUTO, DA, SA, VLANPrio, VLANID, N_SDU)

PDU	Variable	Local variable representing the UDP-RTA-PDU or RTA-PDU depending on transport. Used to show the protocol specific setup and validation of RTA fields. The setup and validation of additional PDU-Parameters (for example IP-header) is not shown. The PDUType is shown in Table 202.
Expected_Seq_Count	Variable	This local variable contains the counter value which shall be used at the receipt of the next DATA PDU.
Expected_Seq_CountO	Variable	This local variable contains the previous counter value of the Seq_Count variable.

4.8.4.4 Remote Service Interface Initiator

4.8.4.4.1 General

The Remote Service Interface Initiator (RSII) fragments and sends requests and receives and defragments responses.

RSII and RSIIN are closely related protocol machines.

4.8.4.4.2 Primitive definitions

4.8.4.4.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by RSII are described in the service definition and shown in Table 287.

Table 287 – Remote primitives issued or received by RSII

Primitive	Source	Destination	Associated parameters	Functions
RSI_Call.req	CMCTL	RSII	ISAP, OpNum, ArgsRspMaxLength, ArgsReqLength, ArgsReq	—
RSI_Call.cnf	RSII	CMCTL RSIIN	ISAP, PNIOStatus, ArgsRspLength, ArgsRsp	—

Primitive	Source	Destination	Associated parameters	Functions
RSI_I_Abort.req	CMCTL	RSII	ISAP, PNIOStatus	—
RSI_I_Abort.cnf	RSII	CMCTL	ISAP, PNIOStatus	—
RSI_R_Abort.ind	RSII	CMCTL	ISAP, PNIOStatus	—
LMPM_R_Data.req	RSII	LMPM	CREP, R_SDU	—
LMPM_R_Data.cnf	LMPM	RSII	CREP	—
LMPM_R_Data.ind	LMPM	RSII	CREP, R_SDU	—

4.8.4.4.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by RSII are described in the service definition and shown in Table 288.

Table 288 – Local primitives issued or received by RSII

Primitive	Source	Destination	Associated parameters	Functions
RSI_I_Add_req	CMCTL	RSII	ISAP, RMAC, VendorID, DeviceID, Instance, InitialSendSeqNum	Add an RSII Instance
RSI_I_Add_cnf	RSII	CMCTL	ISAP, PNIOStatus	—

4.8.4.4.3 State transition diagram

The state transition diagram of the RSII is shown in Figure 93.

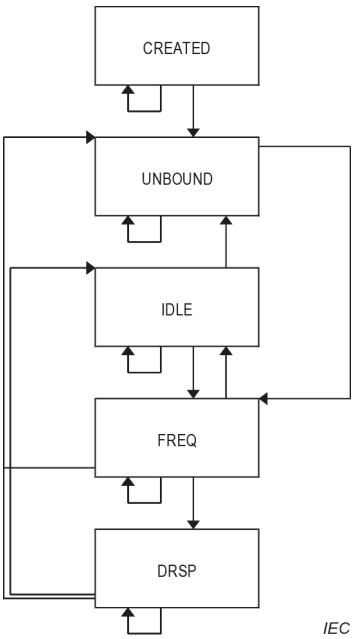


Figure 93 – State transition diagram of RSII

States of the RSII are:

CREATED	Waiting for an add request
UNBOUND	Waiting for a connect request
IDLE	Waiting for the next request
FREQ	Fragment and send RSI request
DRSP	Receive and defragment RSI response

4.8.4.4.4 State machine description

This state machine, addressed by SourceAddress, Source-ServiceAccessPoint, DestinationAddress and Destination-ServiceAccessPoint, handles the R-Data service. This state machine is the initiator of the RSI transport protocol.

A LMPM_R_Data.ind primitive will be accepted if the value of the PDUType.Type field is FREQ-RTA-PDU, FRSP-RTA-PDU, ACK-RTA-PDU or ERR-RTA-PDU and the PDUType.Version is RTA_VERS=2.

4.8.4.4.5 RSII state table

Table 289 contains the description of the RSII state machine.

Table 289 – RSII state table

#	Current State	Event /Condition =>Action	Next State
1	CREATED	**RSI_I_Add_req ()** => /* store data */ NextSeqNum := InitialSendSeqNum SendSeqNum := 0xFFFE RecvSeqNum := 0xFFFE RSI_I_Add_cnf (+)	UNBOUND
2	CREATED	**RSI_Call.req ()** => RSI_Call.cnf (-)	CREATED
3	CREATED	**RSI_I_Abort.req ()** => RSI_I_Abort.cnf (-)	CREATED
4	CREATED	**R_Ack_ind** => ignore	CREATED
5	CREATED	**R_Frsp_ind** => ignore	CREATED
6	CREATED	**R_Error_ind** => ignore	CREATED
7	CREATED	**TimerExpired (ANY)** => ignore	CREATED
8	CREATED	**R_SendAck_cnf** => ignore	CREATED
9	CREATED	**R_SendFreq_cnf** => ignore	CREATED

#	Current State	Event /Condition =>Action	Next State
10	CREATED	**R_SendError_cnf** => ignore	CREATED
11	UNBOUND	**RSI_Call.req ()** /OpNum in [Connect, ReadImplicit, ReadConnectionless, SecurityAssociationControl] => /* init new session */ RSAP := CON-SAP NextSeqNum := MAX(NextSeqNum, SendSeqNum, RecvSeqNum) NextSeqNum += 11 – 1 /* 11 is the next prime after the largest WindowSize */ SendSeqNum := NextSeqNum SendWindowSize := 2 /* smallest WindowSize */ SendCallSequence := 0x07 RecvSeqNum := 0xFFFE RecvAckNum := 0xFFFE RecvWindowSize := 1 /* init call */ TimeoutCount := 0 RecvFragOffset := 0 SendCallSequence += 1 M_PrepareRequest (OpNum, ArgsRspMaxLength, ArgsReqLength, ArgsReq) StartTimer (RemoteApplicationTimeout) SendSeqNum += 1 R_SendFreq_req	FREQ
12	UNBOUND	**RSI_Call.req ()** /OpNum not-in [Connect, ReadImplicit, ReadConnectionless, SecurityAssociationControl] => RSI_Call.cnf (-)	UNBOUND
13	UNBOUND	**RSI_I_Add_req ()** => RSI_I_Add_cnf (-)	UNBOUND
14	UNBOUND	**RSI_I_Abort.req ()** => RSI_I_Abort.cnf (-)	UNBOUND
15	UNBOUND	**R_Ack_ind** => ignore	UNBOUND
16	UNBOUND	**R_Frsp_ind** => ignore	UNBOUND
17	UNBOUND	**R_Error_ind** => ignore	UNBOUND
18	UNBOUND	**TimerExpired (ANY)** => ignore	UNBOUND
19	UNBOUND	**R_SendAck_cnf** => ignore	UNBOUND
20	UNBOUND	**R_SendFreq_cnf** => ignore	UNBOUND
21	UNBOUND	**R_SendError_cnf** => ignore	UNBOUND

#	Current State	Event /Condition =>Action	Next State
22	IDLE	**RSI_Call.req ()** /OpNum not-in [Connect, ReadImplicit, ReadConnectionless, SecurityAssociationControl] => TimeoutCount := 0 RecvFragOffset := 0 SendCallSequence += 1 M_PrepareRequest (OpNum, ArgsRspMaxLength, ArgsReqLength, ArgsReq) StartTimer (RemoteApplicationTimeout) SendSeqNum += 1 R_SendFreq_req	FREQ
23	IDLE	**RSI_Call.req ()** /OpNum in [Connect, ReadImplicit, ReadConnectionless, SecurityAssociationControl] => RSI_Call.cnf (-)	IDLE
24	IDLE	**RSI_I_Add_req ()** => RSI_I_Add_cnf (-)	IDLE
25	IDLE	**RSI_I_Abort.req ()** => ErrorCode2 := abort R_SendError_req RSI_I_Abort.cnf (+)	UNBOUND
26	IDLE	**R_Ack_ind** => ignore	IDLE
27	IDLE	**R_Frsp_ind** => ignore	IDLE
28	IDLE	**R_Error_ind** => RSI_R_Abort.ind	IDLE
29	IDLE	**TimerExpired (ANY)** => ignore	IDLE
30	IDLE	**R_SendAck_cnf** => ignore	IDLE
31	IDLE	**R_SendFreq_cnf** => ignore	IDLE
32	IDLE	**R_SendError_cnf** => ignore	IDLE

33	FREQ	**Do_CheckMoreToSend**
		/M_CheckMoreToSend == LastFragAcked
		=>
		StopTimer (FragTimeout) /* for send */
		StartTimer (FragTimeout) /* for receive */
		TimeoutCount := 0
		if (PDU.Type == FRSP)
		  PDU.TACK := 0
		  R_Frsp_ind /* trigger for new state */
			DRSP

34	FREQ	**Do_CheckMoreToSend** /M_CheckMoreToSend == Send => SendSeqNum += 1 R_SendFreq_req	FREQ

#	Current State	Event /Condition =>Action	Next State
35	FREQ	**Do_CheckMoreToSend** /M_CheckMoreToSend == WaitAck => /* Note: A fragment with TACK or the last fragment was sent, just wait for ACK */	FREQ
36	FREQ	**Do_CheckMoreToSend** /M_CheckMoreToSend == Timeout && OpNum not-in [ReadImplicit, ReadConnectionless] => /* Note: more than MaxFragTimeoutCount Timeouts occur without response */ RSI_Call.cnf (-)	IDLE
37	FREQ	**Do_CheckMoreToSend** /M_CheckMoreToSend == Timeout && OpNum in [ReadImplicit, ReadConnectionless] => /* Note: more than MaxFragTimeoutCount Timeouts occur without response */ RSI_Call.cnf (-)	UNBOUND
38	FREQ	**RSI_I_Add_req ()** => RSI_I_Add_cnf (-)	FREQ
39	FREQ	**RSI_Call.req ()** => ErrorCode2 := state conflict RSI_Call.cnf (-)	FREQ
40	FREQ	**RSI_I_Abort.req ()** => ErrorCode2 := abort RSI_Call.cnf (-) R_SendError_req RSI_I_Abort.cnf (+)	UNBOUND

41	FREQ	**R_Ack_ind**
		=>
		if (RSAP == CON_SAP)
		  RSAP := PDU.SourceSAP
		  RecvWindowSize := PDU.AddFlags.WindowSize
		  Update field sendTack within SendArray according to new RecvWindowSize
		
		RecvAckNum := MAX(RecvAckNum, PDU.AckSeqNum)
		TimeoutCount := 0
		if (PDU.TACK)
		  R_SendAck_req
		
		Do_CheckMoreToSend (Ack)
			FREQ

42	FREQ	**R_Frsp_ind**
		//* SeqNum check not here, see LastFragAcked and DRSP::R_Frsp_ind */
		=>
		/* Note: first handle implicit Ack, then see LastFragAcked */
		if (RSAP == CON_SAP)
		  RSAP := PDU.SourceSAP
		  RecvWindowSize := PDU.AddFlags.WindowSize
		  Update field sendTack within SendArray according to new RecvWindowSize
		
		RecvAckNum := MAX(RecvAckNum, PDU.AckSeqNum)
		TimeoutCount := 0
		if (PDU.TACK)
		  R_SendAck_req
		
		Do_CheckMoreToSend (Ack)
			FREQ

43	FREQ	**R_Error_ind** => StopTimer (ANY) RSI_R_Abort.ind RSI_Call.cnf (-)	IDLE

#	Current State	Event /Condition =>Action	Next State
44	FREQ	**TimerExpired (FragTimeout)** => TimeoutCount += 1 Do_CheckMoreToSend (Timeout)	FREQ
45	FREQ	**TimerExpired (RemoteApplicationTimeout)** => ignore /* never occurs, see FragTimeout */	FREQ
46	FREQ	**R_SendAck_cnf** => StartTimer (FragTimeout)	FREQ
47	FREQ	**R_SendFreq_cnf** => StartTimer (FragTimeout) Do_CheckMoreToSend (SendCnf)	FREQ
48	FREQ	**R_SendError_cnf** => ignore	FREQ
49	DRSP	**Do_LastFrag_ind** /OpNum not-in [ReadImplicit, ReadConnectionless] => StopTimer (ANY) PNIOStatus := KeepPNIOStatus ArgsRspLength := RecvFragOffset – PDUOffset RSI_Call.cnf (+)	IDLE
50	DRSP	**Do_LastFrag_ind** /OpNum in [ReadImplicit, ReadConnectionless] => StopTimer (ANY) PNIOStatus := KeepPNIOStatus ArgsRspLength := RecvFragOffset – PDUOffset RSI_Call.cnf (+)	UNBOUND
51	DRSP	**RSI_I_Add_req ()** => RSI_I_Add_cnf (-)	DRSP
52	DRSP	**RSI_Call.req ()** => RSI_Call.cnf (-)	DRSP
53	DRSP	**RSI_I_Abort.req ()** => StopTimer (ANY) ErrorCode2 := abort RSI_Call.cnf (-) R_SendError_req RSI_I_Abort.cnf (+)	UNBOUND
54	DRSP	**R_Ack_ind** => RecvAckNum := MAX(RecvAckNum, PDU.AckSeqNum) TimeoutCount := 0 if (PDU.TACK) R_SendAck_req	DRSP

#	Current State	Event /Condition =>Action	Next State

55	DRSP	**R_Frsp_ind**
		/PDU.SendSeqNum == (RecvSeqNum == 0xFFFE ? NextSeqNum + 1 :
		RecvSeqNum + 1)
		&& PDU.FragOffset == RecvFragOffset
		&& M_CheckBufferSize
		=>
		/* Expected: do Ack if requested; then do defragmentation */
		RecvAckNum := MAX(RecvAckNum, PDU.AckSeqNum)
		TimeoutCount := 0
		if (isFirstFrag)
		KeepPNIOStatus := PDU.PNIOStatus
		if (PDU.TACK)
		R_SendAck_req
		
		Do_DeFrag_ind
			DRSP

56	DRSP	**R_Frsp_ind** /! (PDU.SendSeqNum == (RecvSeqNum == 0xFFFE ? NextSeqNum + 1 : RecvSeqNum + 1) && PDU.FragOffset == RecvFragOffset && M_CheckBufferSize) => /* Not Expected: do only Ack if requested */ if (PDU.TACK) R_SendAck_req /* do not chain Do_DeFrag_ind */	DRSP
57	DRSP	**Do_DeFrag_ind** => /* Expected: do defragmentation */ ArgsRsp := M_Copy_FragData RecvSeqNum := PDU.SendSeqNum RecvFragOffset += PDU.VarPartLen – 4 if (isLastFrag) Do_LastFrag_ind	DRSP
58	DRSP	**R_Error_ind** => StopTimer (ANY) RSI_R_Abort.ind RSI_Call.cnf (-)	IDLE
59	DRSP	**TimerExpired (FragTimeout)** /! (TimeoutCount >= MaxFragTimeoutCount) => TimeoutCount += 1 R_SendAck_req	DRSP
60	DRSP	**TimerExpired (FragTimeout)** /TimeoutCount >= MaxFragTimeoutCount && OpNum not-in [ReadImplicit, ReadConnectionless] => StopTimer (ANY) RSI_Call.cnf (-)	IDLE
61	DRSP	**TimerExpired (FragTimeout)** /TimeoutCount >= MaxFragTimeoutCount && OpNum in [ReadImplicit, ReadConnectionless] => StopTimer (ANY) RSI_Call.cnf (-)	UNBOUND
62	DRSP	**TimerExpired (RemoteApplicationTimeout)** /OpNum not-in [ReadImplicit, ReadConnectionless] => StopTimer (ANY) RSI_Call.cnf (-)	IDLE

#	Current State	Event /Condition =>Action	Next State
63	DRSP	**TimerExpired (RemoteApplicationTimeout)** /OpNum in [ReadImplicit, ReadConnectionless] => StopTimer (ANY) RSI_Call.cnf (-)	UNBOUND
64	DRSP	**R_SendAck_cnf** => StartTimer (FragTimeout)	DRSP
65	DRSP	**R_SendFreq_cnf** => ignore	DRSP
66	DRSP	**R_SendError_cnf** => ignore	DRSP

4.8.4.4.6 Functions, Macros, Timers and Variables

Table 290 contains the functions, macros, timers and variables used by the RSII, their arguments and their descriptions.

Table 290 – Functions, Macros, Timers and Variables used by the RSII

Name	Type	Function/Meaning
MaxFrag-TimeoutCount	Const	This value determines how often a TimerExpired (FragTimeout) may occur. Default value: 3 Note: see [Protocol::Timeout Frag]
MaxVarPartLen	Const	The constant value 1432, see RTA-PDU::VarPartLen
FragTimeout	Timer	Timeout for retransmission, Timeout for busy indicator. Default value: 2 s Note: see [Protocol::Timeout Frag]
Remote-Application-Timeout	Timer	Timeout for finishing an RSI-Call. Default value: 300 s Note: see [Profile::Remote_Application_Timeout]
TimeoutCount	Value	Counts the TimerExpired events.
StartTimer	Function	This local function is used to start or restart a timer.
StopTimer	Function	This local function is used to stop a timer.
TimerExpired	Function	This local function is issued if a timer expires.
ISAP	Value	This RSI-Initiator SAP, see RSI_I_Add
VendorID	Value	see RSI_I_Add
DeviceID	Value	see RSI_I_Add
Instance	Value	see RSI_I_Add
Initial-SendSeqNum	Value	see RSI_I_Add
RSAP	Value	The RSI-Responder SAP
RMAC	Value	The RSI-Responder MAC, see RSI_I_Add
NextSeqNum	Value	The initial SendSeqNum for the next session
SendSeqNum	Value	The largest SendSeqNum sent; macro R_SendFreq_req use it as "send fragment with this SendSeqNum"
SendWindowSize	Value	The WindowSize of this RSI-Initiator
RecvAckNum	Value	The largest AckSeqNum received
RecvSeqNum	Value	The largest SendSeqNum received

Name	Type	Function/Meaning
RecvFragOffset	Value	The next FragOffset to receive, points to CallBuffer
RecvWindowSize	Value	The tailored WindowSize of the RSI-Responder The WindowSize from the first ACK or from the first fragment of the Connect response is taken, see M_UpdateAckInfo.
SendCall-Sequence	Value	CallSequence; Range (0..7); += 1 does a wrap around
KeepPNIOStatus	Value	Keep PNIOStatus from first FRSP Fragment
PDUOffset	Macro	/* The fragmented part of the "PROFINETIOServiceResPDU" starts at this offset within the RSI-PDU */ PDUOffset := RsiHeaderSize (PDU)
PDU.Type	Macro	PDU.PDUType.Type
PDU.TACK	Macro	PDU.AddFlags.TACK
SendArray	Value	Array of 1 to n request fragment to send SendArray.Length is the count of fragments fragments ::= (sendSeqNum, sendTack, sendMoreFrag, offset, dataLen, Data[])

SearchIndexWith-SendSeqNum	Function	/* Return the index within SendArray with SendArray[idx].SendSeqNum == SendSeqNum */
		for (idx := 0; idx < SendArray.Length; ++idx)
		  if (SendArray[idx].SendSeqNum == SendSeqNum) return idx
		endfor
		return undef /* shall never happen */

Search-LastAcked-SendSeqNum	Function	/* Return the highest acknowledged SendSeqNum within SendArray. If no entry is acknowledged, return the first SendSeqNum. -1 because of +1 in SM */
		for (idx := 0, idx := 0; idx < SendArray.Length; ++idx)
		  if (SendArray[idx].SendSeqNum == RecvAckNum) return RecvAckNum
		endfor
		return SendArray[0].SendSeqNum – 1

build	Macro	MARshal an RSI-REQ-PDU from its arguments

M_Prepare-Request	Macro	/* Prepare the args from RSI_Call.req to an array of FREQ-RTA-PDU to send */
		RSI-REQ-PDU Data := build (OpNum, ArgsRspMaxLength, ArgsReqLength, ArgsReq)
		DataLen := sizeOf (Data)
		FragPartLen := MaxVarPartLen – 4 /*sizeOf FopnumOffset*/
		
		SendArray := []
		ssn := SendSeqNum
		offset := 0
		for (len := 0, i := 0; len < DataLen; len += FragPartLen, i += 1)
		  ssn += 1
		  if (len + FragPartLen < DataLen)
		  sendTack := ((i+1) % RecvWindowSize) == 0 ? 1 : 0
		  SendArray[i] := (ssn, sendTack, sendMoreFrag := 1, offset,
		  FragPartLen, Data[len .. len + FragPartLen – 1])
		  offset += FragPartLen
		  else
		  rest := DataLen – len
		  SendArray[i] := (ssn, sendTack := 0, sendMoreFrag := 0, offset
		  rest, Data[len .. len + rest – 1])
		  endif
		endfor

Do_LastFrag_ind	Event	Signal an internal event. The last fragment is received.
Do_CheckMore-ToSend(Trigger)	Event	Signal an internal event. Trigger is the parameter of M_CheckMoreToSend().
Do_DeFrag_ind	Event	Signal an internal event. Do defragmentation of an FRSP PDU

Name	Type	Function/Meaning

M_CheckMore-ToSend()	Macro	/* SendSeqNum was send; check if there is more to send; return one of: */
		/* – Send: send next FRAG of prepared array */
		/* – WaitAck: wait for ACK, more FRAGs to send or wait for ACK of the last FRAG */
		/* – LastFragAcked: the last FRAG from array was acknowledged */
		/* – Timeout: timeout too often, cancel call */
		idx := SendArray.SearchIndexWithSendSeqNum (SendSeqNum)
		
		if (Trigger == SendCnf)
		if (SendArray[idx].sendMoreFrag == 0) return WaitAck endif
		if (SendArray[idx].sendTack == 1) return WaitAck endif
		return Send
		endif
		if (Trigger == Timeout)
		if (TimeoutCount > MaxFragTimeoutCount) return Timeout endif
		SendSeqNum := SendArray.SearchLastAckedSendSeqNum ()
		return Send
		endif
		if (Trigger == Ack)
		if (SendArray[idx].sendMoreFrag == 0)
		if (SendArray[idx].SendSeqNum == RecvAckNum)
		return LastFragAcked
		endif
		return WaitAck
		elseif (SendArray[idx].sendTack == 1)
		if (SendArray[idx].SendSeqNum == RecvAckNum) return Send endif
		return WaitAck
		endif
		SendSeqNum := SendArray.SearchLastAckedSendSeqNum ()
		return Send
		endif

M_Copy_-FragData	Macro	Copy VarPartLen data from the FRSP-RTA-PDU to ArgsRsp at RecvFragOffset
M_Check-BufferSize	Macro	/* outsourced checks to keep overview */ if ((PDU.VarPartLen + RecvFragOffset) > ArgsRspMaxLength) return false return true
isFirstFrag	Function	PDUType.Type IN (RTA_TYPE_FRSP) && PDU.Foffset == 0
isLastFrag	Function	PDUType.Type IN (RTA_TYPE_FRSP) && AddFlags.MoreFrag == 0
R_SendFreq_req	Macro	/* Send a FREQ-RTA-PDU */ PDU.DA := RMAC idx := SendArray.SearchIndexWithSendSeqNum (SendSeqNum) PDU.DestinationSAP := RSAP PDU.SourceSAP := ISAP PDU.PDUType.Version := 2 PDU.PDUType.Type := FREQ PDU.AddFlags.WindowSize := SendWindowSize PDU.AddFlags.TACK := SendArray[idx].sendTack PDU.AddFlags.MoreFrag := SendArray[idx].sendMoreFrag PDU.AddFlags.Notification := 0 PDU.SendSeqNum := SendSeqNum PDU.AckSeqNum := RecvSeqNum PDU.VarPartLen := SendArray[idx].dataLen + 4 /*sizeOf FopnumOffset*/ PDU.FopnumOffset.Offset := SendArray[idx].offset PDU.FopnumOffset.OpNum := OpNum PDU.FopnumOffset.CallSequence := SendCallSequence PDU.Data := SendArray[idx].Data LMPM_R_Data.req
R_sendFreq_cnf	Macro	LMPM_R_Data.cnf /PDU.PDUType.Type == FREQ

Name	Type	Function/Meaning
R_SendAck_req	Macro	/* Send a RTA-ACK-PDU */ PDU.DA := RMAC PDU.DestinationSAP := RSAP PDU.SourceSAP := ISAP PDU.PDUType.Version := 2 PDU.PDUType.Type := ACK PDU.AddFlags.WindowSize := SendWindowSize PDU.AddFlags.TACK := 0 PDU.AddFlags.MoreFrag := 0 PDU.AddFlags.Notification := 0 PDU.SendSeqNum := SendSeqNum PDU.AckSeqNum := RecvSeqNum PDU.VarPartLen := 0 LMPM_R_Data.req
R_SendAck_cnf	Macro	LMPM_R_Data.cnf /PDU.PDUType.Type == ACK
R_SendError_req	Macro	/* Send a RTA-ERR-PDU */ PDU.DA := RMAC PDU.DestinationSAP := RSAP PDU.SourceSAP := ISAP PDU.PDUType.Version := 2 PDU.PDUType.Type := ERR PDU.AddFlags.WindowSize := SendWindowSize PDU.AddFlags.TACK := 0 PDU.AddFlags.MoreFrag := 0 PDU.AddFlags.Notification := 0 PDU.SendSeqNum := SendSeqNum PDU.AckSeqNum := RecvSeqNum PDU.VarPartLen := 4 PDU.PNIOStatus.ErrorCode := 0xCF PDU.PNIOStatus.ErrorDecode := 0x81 PDU.PNIOStatus.ErrorCode1 := 0xFD PDU.PNIOStatus.ErrorCode2 := ErrorCode2 LMPM_R_Data.req
R_SendError_cnf	Macro	LMPM_R_Data.cnf /PDU.PDUType.Type == ERR
<; <=; >; >=; ==	Operator	Compare two SeqNum, take care of wrap around For valid range see definition of SendSeqNum and AckSeqNum
+	Operator	Add two SeqNum, take care of wrap around For valid range see definition of SendSeqNum and AckSeqNum
MAX	Macro	Take a list of SeqNum, return the largest number. Note: 0xFFFE is the lowest number
R_Ack_ind	Macro	LMPM_R_Data.ind /CheckRSI() == Ack
R_Error_ind	Macro	LMPM_R_Data.ind /CheckRSI() == IndicateError{PNIOStatus}
R_Frsp_ind	Macro	/* Note: if an Response has arrived an R_Frsp_ind is indicated */ /* does an implicit ACK and chains a Do_DeFrag_ind */ LMPM_R_Data.ind /CheckRSI() == Frsp

Name	Type	Function/Meaning

CheckRSI	Function	/* possible returns: Ignore / Frsp / Ack / IndicateError{err} */
		
		/* NOTE: R_SDU is a valid RTA-PDU and a valid RSI-PDU, see LMPM */
		PDU := R_SDU
		
		sap := Rsi-DsapLookup ()
		if (sap == undef || sap != ISAP) return Ignore
		
		if (PDU.SourceMAC != RMAC) return Ignore
		if (RSAP is known && PDU.SourceSAP != RSAP) return Ignore
		
		if (PDU.Type == FREQ) return Ignore
		if (PDU.Type == FRSP) return CheckFrspData ()
		if (PDU.Type == ACK) return CheckAckData ()
		if (PDU.Type == ERR) return CheckErrorData ()
		
		return Ignore

CheckFrspData	Function	if (PDU.AckSeqNum > SendSeqNum)
		return Ignore
		if (PDU.AckSeqNum != 0xFFFE
		&& PDU.AckSeqNum < SendSeqNum – SendWindowSize)
		return Ignore
		return Frsp

CheckAckData	Function	if (PDU.AckSeqNum > SendSeqNum)
		return Ignore
		if (PDU.AckSeqNum != 0xFFFE
		&& PDU.AckSeqNum < SendSeqNum – SendWindowSize)
		return Ignore
		return Ack

CheckErrorData	Function	if (PDU.PNIOStatus == 0) return Ignore return IndicateError{PDU.PNIOStatus}
Rsi-DsapLookup	Function	sap := Search PDU.DestinationSAP within RSII::ISAP if (sap not found) return undef return sap

4.8.4.5 Remote Service Interface Initiator Notification

4.8.4.5.1 Primitive definitions

4.8.4.5.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by RSIIN are described in the service definition and shown in Table 291.

Table 291 – Remote primitives issued or received by RSIIN

Primitive	Source	Destination	Associated parameters	Functions
RSI_Notification.ind	RSIIN	CTLRSI	ISAP, Mode	—
RSI_Call.cnf	RSII	RSIIN	ISAP, PNIOStatus, ArgsRspLength, ArgsRsp	—

4.8.4.5.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by RSIIN are described in the service definition and shown in Table 292.

Table 292 – Local primitives issued or received by RSIIN

Primitive	Source	Destination	Associated parameters	Functions
R_Frsp_ind	RSII	RSIIN	ISAP, PDU	—
R_Ack_ind	RSII	RSIIN	ISAP, PDU	—

4.8.4.5.2 State transition diagram

The state transition diagram of the RSIIN is shown in Figure 94.

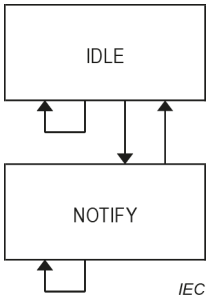


Figure 94 – State transition diagram of RSIIN

States of the RSIIN are:

IDLE The notification flag is not set

NOTIFY The notification flag is set

4.8.4.5.3 State machine description

The RSI initiator notification state machine (RSIIN) sets the notification flag and informs the upper state machine. The flag is reset when the notification has been read.

RSIIN and RSII are closely related protocol machines.

4.8.4.5.4 RSIIN state table

Table 293 contains the description of the RSIIN state machine.

Table 293 – RSIIN state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**R_Ack_ind** /AddFlags.Notification == 1 => Mode := Set RSI_Notification.ind	NOTIFY
2	IDLE	**R_Ack_ind** /AddFlags.Notification != 1 => ignore	IDLE
3	IDLE	**R_Frsp_ind** /AddFlags.Notification == 1 => Mode := Set RSI_Notification.ind	NOTIFY
4	IDLE	**R_Frsp_ind** /AddFlags.Notification != 1 => ignore	IDLE
5	IDLE	**RSI_Call.cnf** => ignore	IDLE
6	NOTIFY	**RSI_Call.cnf** /PNIOStatus == OK && OpNum == ReadNotification /* ApplReadyBlock */ => Mode := Reset	IDLE
7	NOTIFY	**RSI_Call.cnf** /! (PNIOStatus == OK && OpNum == ReadNotification) => ignore	NOTIFY
8	NOTIFY	**R_Ack_ind** => ignore	NOTIFY
9	NOTIFY	**R_Frsp_ind** => ignore	NOTIFY

4.8.4.5.5 Functions, Macros, Timers and Variables

Table 294 contains the functions, macros, timers and variables used by the RSIIN, their arguments and their descriptions.

Table 294 – Functions, Macros, Timers and Variables used by the RSIIN

Name	Type	Function/Meaning
—	—	—

4.8.4.6 Remote Service Interface Responder

4.8.4.6.1 General

The Remote Service Interface Responder (RSIR) receives and defragments requests and fragments and sends responses.

RSIR and RSIRN are closely related protocol machines.

4.8.4.6.2 Primitive definitions

4.8.4.6.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by RSIR are described in the service definition and shown in Table 295.

Table 295 – Remote primitives issued or received by RSIR

Primitive	Source	Destination	Associated parameters	Functions
RSI_Call.ind	RSIR	CMRSI	RSAP, OpNum, ArgsRspMax, ArgsReqLength, ArgsReq	—
RSI_Call.rsp	CMRSI	RSIR	RSAP, PNIOStatus, ArgsRspLength, ArgsRsp	—
RSI_R_Abort.req	FSPM	RSIR	RSAP, PNIOStatus	—
RSI_R_Abort.cnf	RSIR	FSPM	RSAP	—
RSI_I_Abort.ind	RSIR	CMRSI	RSAP, PNIOStatus	—
LMPM_R_Data.req	RSIR	LMPM	CREP, R_SDU	—
LMPM_R_Data.cnf	LMPM	RSIR	CREP	—
LMPM_R_Data.ind	LMPM	RSIR	CREP, R_SDU	—

4.8.4.6.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by RSIR are described in the service definition and shown in Table 296.

Table 296 – Local primitives issued or received by RSIR

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.8.4.6.3 State transition diagram

The state transition diagram of the RSIR is shown in Figure 95.

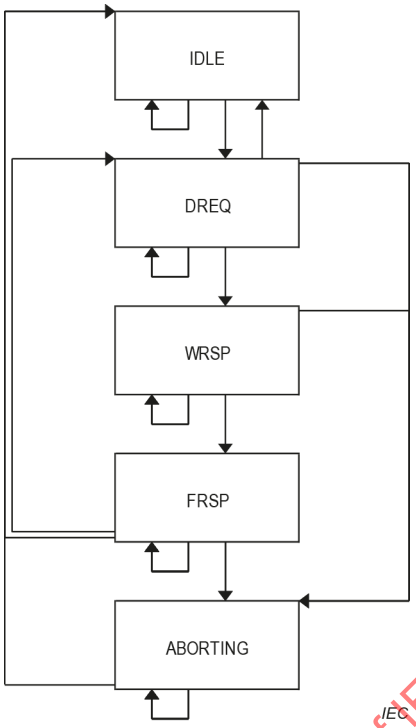


Figure 95 – State transition diagram of RSIR

States of the RSIR are:

IDLE	Waiting for connection establishment
DREQ	Receive and defragment request
WRSP	A RsiCall.ind is issued, waiting for the RsiCall.rsp
FRSP	Fragment and send response
ABORTING	Preparing for disconnection

4.8.4.6.4 State machine description

This state machine, addressed by SourceAddress, Source-ServiceAccessPoint, DestinationAddress and Destination-ServiceAccessPoint, handles the R-Data service. This state machine is the responder of the RSI transport protocol.

A LMPM_R_Data.ind primitive will be accepted if the value of the PDUType.Type field is FREQ-RTA-PDU, FRSP-RTA-PDU, ACK-RTA-PDU or ERR-RTA-PDU and the PDUType.Version is RTA_VERS=2.

4.8.4.6.5 RSIR state table

Table 297 contains the description of the RSIR state machine.

Table 297 – RSIR state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**Do_LocalConnect_ind** => KeepConnectSendSeqNum := PDU.SendSeqNum SendSeqNum := 0xFFFE SendWindowSize := 2 /* smallest WindowSize */ RecvAckNum := 0xFFFE RecvSeqNum :=KeepConnectSendSeqNum – 1 RecvFragOffset := 0 RecvWindowSize := PDU.AddFlags.WindowSize KeepOpNum := invalid ConState := IDLE M_do_fullyBound StartTimer (Timeout) R_Freq_ind /* trigger for new state */	DREQ
2	IDLE	**R_Connect_ind** => Do_LocalConnect_ind	IDLE
3	IDLE	**R_Ack_ind** => ignore	IDLE
4	IDLE	**R_Freq_ind** => ignore	IDLE
5	IDLE	**RSI_Call.rsp ()** => ignore	IDLE
6	IDLE	**R_SendFrsp_cnf** => ignore	IDLE
7	IDLE	**R_SendError_cnf** => ignore	IDLE
8	IDLE	**R_SendAck_cnf** => ignore	IDLE
9	IDLE	**TimerExpired (Timeout)** => ignore	IDLE
10	IDLE	**R_Error_ind** => ignore	IDLE
11	IDLE	**RSI_R_Abort.req ()** => RSI_R_Abort.cnf (+)	IDLE
12	DREQ	**Do_LastFrag_ind** => StopTimer (Timeout) if (ConState == IDLE && KeepConnectSendSeqNum != PDU.SendSeqNum) ConState := ABORT /* Note: more then one fragment */ ArgsReqLength := RecvFragOffset – PDUOffset OpNum := KeepOpNum ArgsRspMax := KeepArgsRspMax RSI_Call.ind ()	WRSP

#	Current State	Event /Condition =>Action	Next State
13	DREQ	**R_Connect_ind** => /* Note: fragmented connect rebuild from scratch */ M_do_halfBound Do_LocalConnect_ind	IDLE
14	DREQ	**R_Ack_ind** => RecvAckNum := MAX (RecvAckNum, PDU.AckSeqNum) if (PDU.TACK) R_SendAck_req	DREQ
15	DREQ	**R_Freq_ind** /PDU.SendSeqNum == (RecvSeqNum + 1) && PDU.FragOffset == RecvFragOffset && M_MoreFreqCheck => /* Expected frag: do ACK if requested; then do defragmentation */ /* UseCase: First FREQ of Connect has TACK set */ RecvSeqNum := PDU.SendSeqNum RecvAckNum := MAX (RecvAckNum, PDU.AckSeqNum) if (PDU.TACK) R_SendAck_req /* Send RSAP to Initiator */ Do_DeFrag_ind	DREQ
16	DREQ	**R_Freq_ind** /! (PDU.SendSeqNum == (RecvSeqNum + 1) && PDU.FragOffset == RecvFragOffset && M_MoreFreqCheck) => /* Not expected frag: do ACK only if requested */ /* Note: will never happen with R_Connect_ind */ if (PDU.TACK) R_SendAck_req /* do not chain Do_DeFrag_ind */	DREQ

17	DREQ	**Do_DeFrag_ind**
		=>
		/* Expected frag: do defragmentation */
		ArgsReq := M_Copy_FragData
		RecvFragOffset := RecvFragOffset + PDU.VarParLen – 4
		
		if (isFirstFrag)
		  KeepOpNum := PDU.FopNum
		  KeepArgsRspMax := PDU.RspMaxLength
		  KeepCallSequence := PDU.FcallSequence
		  if (KeepArgsRspMax > CallBuffer.MaxLength)
		  KeepArgsRspMax = CallBuffer.MaxLength
		  /* all possible server rsp shall fit into this size */
		
		StartTimer (Timeout) /* restart on every frag received */
		if (isLastFrag)
		  Do_LastFrag_ind
			DREQ

18	DREQ	**RSI_Call.rsp ()** => ignore	DREQ
19	DREQ	**R_SendFrsp_cnf** => ignore	DREQ
20	DREQ	**R_SendError_cnf** => ignore	DREQ
21	DREQ	**R_SendAck_cnf** => ignore	DREQ

#	Current State	Event /Condition =>Action	Next State
22	DREQ	**TimerExpired (Timeout)** => /* Note: RSII does not send all fragments */ RSI_I_Abort.ind ()	ABORTING
23	DREQ	**R_Error_ind** => RSI_I_Abort.ind ()	ABORTING
24	DREQ	**RSI_R_Abort.req ()** => R_SendError_req	ABORTING
25	WRSP	**RSI_Call.rsp ()** /Check_RSI_Call_rsp == OK => if (SendSeqNum == 0xFFFE) SendSeqNum := KeepConnectSendSeqNum – 1 M_PrepareResponse (PNIOStatus, ArgsRsp, ArgsRspLength) SendSeqNum := SendSeqNum + 1 R_SendFrsp_req	FRSP
26	WRSP	**RSI_Call.rsp ()** /Check_RSI_Call_rsp == RspMaxLength => RSI_I_Abort.ind ()	ABORTING
27	WRSP	**R_Connect_ind** => /* Note: connection monitoring */ R_Ack_ind	WRSP
28	WRSP	**R_Ack_ind** => RecvAckNum := MAX (RecvAckNum, PDU.AckSeqNum) if (PDU.TACK || PDU.Type == ACK || (PDU.Type == FREQ && isLastFrag)) R_SendAck_req /* busy indicator */	WRSP
29	WRSP	**R_Freq_ind** => /* Note: connection monitoring */ R_Ack_ind	WRSP
30	WRSP	**R_SendFrsp_cnf** => ignore	WRSP
31	WRSP	**R_SendError_cnf** => ignore	WRSP
32	WRSP	**R_SendAck_cnf** => ignore	WRSP
33	WRSP	**TimerExpired (Timeout)** => ignore	WRSP
34	WRSP	**R_Error_ind** => RSI_I_Abort.ind ()	ABORTING
35	WRSP	**RSI_R_Abort.req ()** => R_SendError_req	ABORTING

#	Current State	Event /Condition =>Action	Next State
36	FRSP	**Do_CheckMoreToSend** /M_CheckMoreToSend == LastFragAcked => /* Note: prepare next call */ ConState := ABORT RecvFragOffset := 0 if (PDU.Type == FREQ) Do_DeFrag_ind /* trigger for new state */	DREQ
37	FRSP	**Do_CheckMoreToSend** /M_CheckMoreToSend == Send => SendSeqNum := SendSeqNum + 1 R_SendFrsp_req	FRSP
38	FRSP	**Do_CheckMoreToSend** /M_CheckMoreToSend == WaitAck => ignore	FRSP
39	FRSP	**R_Connect_ind** /KeepConnectSeqNum == PDU.SendSeqNum – 1 && KeepOpNum == PDU.FopNum => /* Note: connection monitoring */ R_Ack_ind	FRSP
40	FRSP	**R_Connect_ind** /KeepConnectSeqNum == PDU.SendSeqNum – 1 && KeepOpNum != PDU.FopNum => ignore	FRSP
41	FRSP	**R_Connect_ind** /KeepConnectSeqNum != PDU.SendSeqNum – 1 && KeepOpNum not-in [ReadImplicit, ReadConnectionless] => /* Note: the abort of the RSII seems missed, abort now */ RSI_I_Abort.ind ()	ABORTING
42	FRSP	**R_Connect_ind** /KeepConnectSeqNum != PDU.SendSeqNum – 1 && KeepOpNum in [ReadImplicit, ReadConnectionless] => /* Note: reuse this instance */ M_do_halfBound Do_LocalConnect_ind	IDLE
43	FRSP	**R_Ack_ind** => RecvAckNum := MAX (RecvAckNum, PDU.AckSeqNum) SendSeqNum := M_PrepareReSend if (PDU.TACK) R_SendAck_req Do_CheckMoreToSend	FRSP
44	FRSP	**R_Freq_ind** /PDU.SendSeqNum == (RecvSeqNum + 1) && PDU.FragOffset == RecvFragOffset && M_MoreFreqCheck => /* Expected: do implicit ACK of the previous call */ /* then do the next call (see LastFragAcked) */ RecvSeqNum := PDU.SendSeqNum RecvAckNum := MAX (RecvAckNum, PDU.AckSeqNum) if (PDU.TACK) R_SendAck_req Do_CheckMoreToSend	FRSP

#	Current State	Event /Condition =>Action	Next State

45	FRSP	**R_Freq_ind**
		/! (PDU.SendSeqNum == (RecvSeqNum + 1)
		&& PDU.FragOffset == RecvFragOffset
		&& M_MoreFreqCheck)
		=>
		/* Not expected: do implicit ACK if requested, check resend */
		
		RecvAckNum := MAX (RecvAckNum, PDU.AckSeqNum)
		
		if (PDU.TACK || isLastFrag)
		SendSeqNum := M_PrepareReSend
		if (PDU.TACK)
		R_SendAck_req
		
		Do_CheckMoreToSend
			FRSP

46	FRSP	**RSI_Call.rsp ()** => ignore	FRSP
47	FRSP	**R_SendFrsp_cnf** => if (KeepOpNum in [ReadImplicit, ReadConnectionless]) StartTimer (Timeout) Do_CheckMoreToSend	FRSP
48	FRSP	**R_SendError_cnf** => ignore	FRSP
49	FRSP	**R_SendAck_cnf** => ignore	FRSP
50	FRSP	**TimerExpired (Timeout)** /OpNum not-in [ReadImplicit, ReadConnectionless] => ignore /* MISS shall occur */	FRSP
51	FRSP	**TimerExpired (Timeout)** /OpNum in [ReadImplicit, ReadConnectionless] => M_do_halfBound	IDLE
52	FRSP	**R_Error_ind** => RSI_I_Abort.ind ()	ABORTING
53	FRSP	**RSI_R_Abort.req ()** => R_SendError_req	ABORTING
54	ABORTING	**R_SendError_cnf** => M_do_halfBound RSI_R_Abort.cnf ()	IDLE
55	ABORTING	**R_Connect_ind** => ignore	ABORTING
56	ABORTING	**R_Ack_ind** => ignore	ABORTING
57	ABORTING	**R_Freq_ind** => ignore	ABORTING
58	ABORTING	**RSI_Call.rsp ()** => ignore	ABORTING
59	ABORTING	**R_SendFrsp_cnf** => ignore	ABORTING

#	Current State	Event /Condition =>Action	Next State
60	ABORTING	**R_SendAck_cnf** => ignore	ABORTING
61	ABORTING	**TimerExpired (Timeout)** => ignore	ABORTING
62	ABORTING	**R_Error_ind** => ignore	ABORTING
63	ABORTING	**RSI_R_Abort.req ()** => R_SendError_req	ABORTING

4.8.4.6.6 Functions, Macros, Timers and Variables

Table 298 contains the functions, macros, timers and variables used by the RSIR, their arguments and their descriptions.

Table 298 – Functions, Macros, Timers and Variables used by the RSIR

Name	Type	Function/Meaning
Timeout	Timer	Timeout for partial request detection Default value: 4 * RSII::FragTimeout
StartTimer	Function	This local function is used to start or restart a timer.
StopTimer	Function	This local function is used to stop a timer.
TimerExpired	Function	This local function is issued if a timer expires.
RSAP	Value	This RSI-Responder SAP, see RSIRN::RSI_R_Add
Binding	Value	see RSIRN::RSI_R_Add
VendorID	Value	see RSIRN::RSI_R_Add
DeviceID	Value	see RSIRN::RSI_R_Add
Instance	Value	see RSIRN::RSI_R_Add
CallBuffer	Value	see RSIRN::RSI_R_Add
ISAP	Value	The RSI-Initiator SAP
IMAC	Value	The RSI-Initiator MAC
SendSeqNum	Value	The largest SendSeqNum sent
SendWindowSize	Value	The WindowSize of this RSI-Responder
RecvAckNum	Value	The largest AckSeqNum received
RecvSeqNum	Value	The largest SendSeqNum received, the next PDU to receive shall have a SendSeqNum equal to RecvSeqNum + 1
RecvFragOffset	Value	The next FragOffset to receive, offset of CallBuffer
RecvWindowSize	Value	The tailored WindowSize of the RSI-Initiator /* The WindowSize within the first Connect fragment will be taken */ /* Tailoring to a lower value is implementation specific */
KeepConnectSend SeqNum	Value	Keep the SendSeqNum from the Connect fragment
SendArray	Value	Array of 1 to n Response fragments to send. SendArray.Length is the count of fragments fragments ::= (sendSeqNum, sendTack, sendMoreFrag, offset, dataLen, Data[])
MaxVarPartLen	Const	The constant value 1432, see RTA-PDU::VarPartLen
M_do_fullyBound	Macro	(IMAC, ISAP) := (PDU.SourceMAC, PDU.SourceSAP)

Name	Type	Function/Meaning
M_do_halfBound	Macro	(IMAC, ISAP) := (undef, undef)
isFirstFrag	Condition	PDU.Type == RTA_TYPE_FREQ && PDU.Foffset == 0
isLastFrag	Condition	PDU.Type == RTA_TYPE_FREQ && AddFlags.MoreFrag == 0
PDUOffset	Macro	/* The fragmented part of the "PROFINETIOServiceReqPDU" starts at this Offset within the RSI-PDU */ PDUOffset := RsiHeaderSize (PDU)
KeepOpNum	Value	Keep PDU Data
KeepArgsRspMax	Value	Keep PDU Data
KeepCallSequence	Value	Keep PDU Data

M_MoreFreqCheck	Macro	/* outsourced checks to keep overview */
		if (! isFirstFrag)
		  if (KeepOpNum != PDU.FopNum) return false endif
		endif
		curr_len := RecvFragOffset + PDU.VarPartLen – 4/*FopnumOffset*/
		if (curr_len > CallBuffer.MaxLength) return false /* doesn't fit in CallBuffer */
		if (curr_len > KeepArgsRspMax) return false /* protocol error: at least Length(RSI-REQ-PDU) */
		return true

M_Copy_FragData	Macro	Copy VarPartLen data from the FREQ-RTA-PDU to the CallBuffer at Foffset
build	Macro	MARshal a RSI-RES-PDU from its arguments

M_Prepare-Response	Macro	/* Prepare the response from RSI_Call.rsp to an array of FRAG-RTA-PDU to send */
		RSI-RES-PDU Data := build (PNIOStatus, ArgsRspLength, ArgsRsp)
		DataLen := sizeOf (Data)
		FragPartLen := MaxVarPartLen – 4 /*sizeOf FopnumOffset*/
		
		SendArray := []
		ssn := SendSeqNum;
		offset := 0
		for (len = 0, i = 0; len < DataLen; len += FragPartLen, i += 1)
		  ssn += 1
		  if (len + FragPartLen < DataLen)
		  sendTack := ((i+1) % RecvWindowSize) == 0 ? 1 : 0
		  SendArray[i] := (ssn, sendTack, sendMoreFrag := 1, offset,
		  FragPartLen, Data[len .. len + FragPartLen – 1])
		  offset += FragPartLen
		  else
		  rest := DataLen – len
		  SendArray[i] := (ssn, sendTack := 0, sendMoreFrag := 0, offset,
		  rest, Data[len .. len + rest – 1])
		  endif
		endfor

M_PrepareReSend	Macro	/* Received an ACK, prepare to ReSend the array of FRAG-RTA-PDU */
		
		for (i = 0; i < SendArray.Length; i += 1)
		  if (SendArray[i].SendSeqNum > RecvAckNum)
		  return SendArray[i].SendSeqNum
		endfor
		return SendSeqNum

M_Check-MoreToSend	Macro	/* check if there is more to send, return one of: */
		/* – Send: send next FRAG of prepared array */
		/* – WaitAck: wait for ACK, more FRAGs to send or wait for ACK of the last FRAG */
		/* – LastFragAcked: the last FRAG was acknowledged (implicit or explicit) */
		
		idx := SendArray.SearchIndexWithSendSeqNum (SendSeqNum)
		
		if (SendArray[idx].sendMoreFrag == 0) /* last frag */
		  if (SendArray[idx].SendSeqNum <= RecvAckNum) return LastFragAcked
		  return WaitAck
		
		if (SendArray[idx].sendTack == 1) /* wait for ack */
		  if (SendArray[idx].SendSeqNum > RecvAckNum) return WaitAck
		  return Send
		
		return Send

Name	Type	Function/Meaning

SearchIndex-WithSendSeqNum	Function	/* return the index within SendArray */
		
		for (idx := 0; idx < SendArray.Length; ++idx)
		  if (SendArray[idx].SendSeqNum == SendSeqNum) return idx
		endfor
		return undef /* shall never happen */

Do_CheckMore-ToSend	Event	Signal an internal event. Check the condition again.
Do_LastFrag_ind	Event	Signal an internal event. Do additional actions on the last fragment.
R_SendFrsp_req	Macro	/* Send a FRSP-RTA-PDU */ idx := SendArray.SearchIndexWithSendSeqNum (SendSeqNum) PDU.DA := IMAC PDU.DestinationSAP := ISAP PDU.SourceSAP := RSAP PDU.PDUType.Version := 2 PDU.PDUType.Type := FRSP PDU.AddFlags.WindowSize := SendWindowSize PDU.AddFlags.TACK := SendArray[idx].sendTack PDU.AddFlags.MoreFrag := SendArray[idx].sendMoreFrag PDU.AddFlags.Notification := (RSI-R-Notification.state == NOTIFY) ? 1 : 0 PDU.SendSeqNum := SendSeqNum PDU.AckSeqNum := RecvSeqNum PDU.VarPartLen := SendArray[idx].dataLen + 4 /*sizeOf FopnumOffset*/ PDU.FopnumOffset.Offset := SendArray[idx].offset PDU.FopnumOffset.OpNum := KeepOpNum PDU.FopnumOffset.CallSequence := KeepCallSequence PDU.Data := SendArray[idx].Data LMPM_R_Data.req
R_SendFrsp_cnf	Macro	LMPM_R_Data.cnf /PDU.PDUType.Type == FRSP
R_SendAck_req	Macro	/* Send a RTA-ACK-PDU */ PDU.DA := IMAC PDU.DestinationSAP := ISAP PDU.SourceSAP := RSAP PDU.PDUType.Version := 2 PDU.PDUType.Type := ACK PDU.AddFlags.WindowSize := SendWindowSize PDU.AddFlags.TACK := 0 PDU.AddFlags.MoreFrag := 0 PDU.AddFlags.Notification := (RSIRN.state == NOTIFY) ? 1 : 0 PDU.SendSeqNum := SendSeqNum PDU.AckSeqNum := RecvSeqNum PDU.VarPartLen := 0 LMPM_R_Data.req
R_SendAck_cnf	Macro	LMPM_R_Data.cnf /PDU.PDUType.Type == ACK

Name	Type	Function/Meaning
R_SendError_req	Macro	/* Send a RTA-ERR-PDU */ PDU.DA := IMAC PDU.DestinationSAP := ISAP PDU.SourceSAP := RSAP PDU.PDUType.Version := 2 PDU.PDUType.Type := ERR PDU.AddFlags.WindowSize := SendWindowSize PDU.AddFlags.TACK := 0 PDU.AddFlags.MoreFrag := 0 PDU.AddFlags.Notification := 0 PDU.SendSeqNum := SendSeqNum PDU.AckSeqNum := RecvSeqNum PDU.VarPartLen := 4 PDU.PNIOStatus.ErrorCode := 0xCF PDU.PNIOStatus.ErrorDecode := 0x81 PDU.PNIOStatus.ErrorCode1 := 0xFD PDU.PNIOStatus.ErrorCode2 := ErrorCode2 LMPM_R_Data.req
R_SendError_cnf	Macro	LMPM_R_Data.cnf /PDU.PDUType.Type == ERR
FopNum	Macro	FopnumOffset.Opnum
Foffset	Macro	FopnumOffset.Offset
FcallSequence	Macro	FopnumOffset.CallSequence
PDU.Type	Macro	PDU.PDUType.Type
PDU.TACK	Macro	PDU.AddFlags.TACK
<; <=; >; >=	Operator	Compare two SeqNum, take care of wrap around For valid range see definition of SendSeqNum and AckSeqNum
+	Operator	Add two SeqNum, take care of wrap around For valid range see definition of SendSeqNum and AckSeqNum
MAX	Operator	Return the larger of two SeqNum, take care of wrap around For valid range see definition of SendSeqNum and AckSeqNum
R_Error_ind	Macro	/* Note: an Error PDU is received */ LMPM_R_Data.ind /CheckRSI() == IndicateError => PNIOStatus := IndicateError.err /* Note: if a ConnectReRun with ConState Abort is received, an error with PNIOStatus ReRun is indicated */ LMPM_R_Data.ind /CheckRSI() == ConnectReRun && ConState == ABORT => PNIOStatus := ReRun
R_Ack_ind	Macro	LMPM_R_Data.ind /CheckRSI() == Ack
R_Freq_ind	Macro	/* Note: if a Request is received; chain a Do_DeFrag_ind */ LMPM_R_Data.ind /CheckRSI() == Freq
R_Connect_ind	Macro	/* Note: if a Connect is received, an R_Connect_ind is indicated. */ /* This indication may chain an R_Freq_ind and a Do_DeFrag_ind */ LMPM_R_Data.ind /CheckRSI() == ConnectFreq /* Note: if a ConnectReRun with ConState Idle is received, an R_Connect_ind is indicated. */ /* This indication may chain an R_Freq_ind and a Do_DeFrag_ind */ LMPM_R_Data.ind /CheckRSI() == ConnectReRun && ConState != ABORT

Name	Type	Function/Meaning

CheckRSI	Function	/* Possible returns: */
		/* – Ignore */
		/* – IndicateError{err} .. event containing additional information err */
		/* – RespondError{err} .. used by RSIRN */
		/* – Ack */
		/* – Freq */
		/* – ConnectFreq */
		/* – ConnectReRun */
		
		/* NOTE: R_SDU is a valid RTA-PDU and a valid RSI-PDU, see LMPM */
		PDU := R_SDU
		
		sap := Rsi-DsapLookup ()
		if (sap == undef) return CheckFreqConnect ()
		
		if (PDU.SourceMAC != IMAC) return Ignore
		if (PDU.SourceSAP != ISAP) return Ignore
		
		if (PDU.PDUType == FREQ) return CheckFreqData ()
		if (PDU.PDUType == FRSP) return Ignore
		if (PDU.PDUType == ACK) return CheckAckData ()
		if (PDU.PDUType == ERR) return CheckErrorData ()
		
		return Ignore

CheckFreqConnect	Function	if (PDU.DestinationSAP != CON-SAP) return Ignore
		if (PDU.PDUType != FREQ) return Ignore
		if (PDU.SendSeqNum not-in [0..0x7FFF) return Ignore
		if (PDU.AckSeqNum != 0xFFFE) return Ignore
		if (PDU.Foffset != 0) return Ignore
		
		if (FopNum in [Connect, ReadConnectionless, SecurityAssociationControl])
		if ((rsap := Rsi-SsapLookup(AR)) != undef)
		return ConnectReRun(rsap)
		if (! Rsi-InterfaceAvailable(AR))
		return Ignore
		if (! Rsi-AllocPossible(AR))
		return RespondError{OutOfResources}
		return ConnectFreq
		
		if (FopNum == ReadImplicit)
		if ((rsap := Rsi-SsapLookup(RI)) != undef)
		return ConnectReRun(rsap)
		if (! Rsi-InterfaceAvailable(RI))
		return Ignore
		if (! Rsi-AllocPossible(RI))
		return RespondError{OutOfResources}
		return ConnectFreq
		
		return Ignore

CheckFreqData	Function	if (PDU.AckSeqNum > SendSeqNum) return Ignore
		if (PDU.AckSeqNum != 0xFFFE
		&& PDU.AckSeqNum < SendSeqNum – SendWindowSize)
		return Ignore
		return Freq

CheckAckData	Function	if (PDU.AckSeqNum > SendSeqNum)
		return Ignore
		if (PDU.AckSeqNum != 0xFFFE
		&& PDU.AckSeqNum < SendSeqNum – SendWindowSize)
		return Ignore
		return Ack

CheckErrorData	Function	if (PDU.PNIOStatus == 0) return Ignore return IndicateError{PDU.PNIOStatus}
Rsi-DsapLookup	Function	rsap := Search PDU.DestinationSAP within RSIR::rsap return rsap if found else undef
Rsi-SsapLookup	Function	rsap := Search PDU.SourceMAC, PDU.SourceSAP within RSIR::imac, RSIR:isap. Return rsap if found else undef
Rsi-InterfaceAvailable(Binding_)	Function	True if Binding_, PDU...VendorID, PDU...DeviceID, PDU...Instance exists in any RSIR

Name	Type	Function/Meaning
Rsi-AllocPossible(Binding_)	Function	Searching a free RSIR instance. Binding_, PDU...VendorID, PDU...DeviceID, PDU...Instance must match. RSIR instance shall be in state IDLE. if KeepOpNum in [ReadImplicit, ReadConnectionless] the state FRSP is possible too. if more than one free instance exist, the instance which previously used IMAC and ISAP shall be chosen. return True if found else False
Do_LocalConnect_ind	Event	Signal an internal event. Prepare for a new connect indication.
ConState	Value	Track the state for the connect indication, init with IDLE { IDLE, ABORT }
Do_DeFrag_ind	Event	Signal an internal event. Do defragmentation of an FREQ PDU.
Check_RSI_Call_rsp	Function	Check the RSI_Call.rsp and map to an RSI error.

4.8.4.7 Remote Service Interface Responder Notification

4.8.4.7.1 Primitive definitions

4.8.4.7.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by RSIRN are described in the service definition and shown in Table 299.

Table 299 – Remote primitives issued or received by RSIRN

Primitive	Source	Destination	Associated parameters	Functions
RSI_R_Add_req	CMDEV	RSIRN	RSAP, Binding, VendorID, DeviceID, Instance, CallBuffer	—
RSI_R_Add_cnf	RSIRN	CMDEV	RSAP, PNIOStatus	—
RSI_Notification.req	CMRSI	RSIRN	RSAP, Mode { Set, Reset }	—
RSI_Notification.cnf	RSIRN	CMRSI	RSAP	—
LMPM_R_Data.req	RSIRN	LMPM	CREP, R_SDU	—
LMPM_R_Data.cnf	LMPM	RSIRN	CREP	—
LMPM_R_Data.ind	LMPM	RSIRN	CREP, R_SDU	—

4.8.4.7.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by RSIRN are described in the service definition and shown in Table 300.

Table 300 – Local primitives issued or received by RSIRN

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.8.4.7.2 State transition diagram

The state transition diagram of the RSIRN is shown in Figure 96.

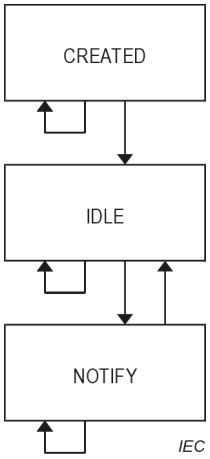


Figure 96 – State transition diagram of RSIRN

States of the RSIRN are:

CREATED	Waiting for add request
IDLE	The notification flag is not set
NOTIFY	The notification flag is set, resend on timeout.

4.8.4.7.3 State machine description

The RSI responder notification state machine (RSIRN) is responsible for the RSIR resources. Additionally, RSIRN sets and resets the notification flag and sends the Notification PDU.

RSIRN and RSIR are closely related protocol machines.

4.8.4.7.4 RSIRN state table

Table 301 contains the description of the RSIRN state machine.

Table 301 – RSIRN state table

#	Current State	Event /Condition =>Action	Next State
1	CREATED	**RSI_R_Add_req ()** => /* store arguments */ CREP.RSIR := create (RSIR) RSI_R_Add_cnf (+)	IDLE
2	CREATED	**RSI_Notification.req** => RSI_Notification.cnf (-)	CREATED
3	CREATED	**TimerExpired (NotifyTimeout)** => ignore	CREATED
4	CREATED	**R_SendNotify_cnf** => ignore	CREATED

#	Current State	Event /Condition =>Action	Next State
5	CREATED	**LMPM_R_Data.ind** => ignore	CREATED
6	CREATED	**R_SendError_cnf** => ignore	CREATED
7	IDLE	**RSI_Notification.req** /Mode == Set => StartTimer (NotifyTimeout) AddFlags.Notification := 1 RSI_Notification.cnf R_SendNotify_req	NOTIFY
8	IDLE	**RSI_Notification.req** /Mode != Set => RSI_Notification.cnf(-)	IDLE
9	IDLE	**RSI_R_Add_req ()** => RSI_R_Add_cnf (-)	IDLE
10	IDLE	**TimerExpired (NotifyTimeout)** => ignore	IDLE
11	IDLE	**R_SendNotify_cnf** => ignore	IDLE
12	IDLE	**LMPM_R_Data.ind** /CheckRSI() == RespondError => /* send back binding issues */ R_SendError_req	IDLE
13	IDLE	**LMPM_R_Data.ind** /CheckRSI() != RespondError => ignore	IDLE
14	IDLE	**R_SendError_cnf** => ignore	IDLE
15	NOTIFY	**RSI_Notification.req** /Mode == Reset => AddFlags.Notification := 0 StopTimer (NotifyTimeout) RSI_Notification.cnf	IDLE
16	NOTIFY	**RSI_Notification.req** /Mode != Reset => RSI_Notification.cnf(-)	NOTIFY
17	NOTIFY	**RSI_R_Add_req ()** => RSI_R_Add_cnf (-)	NOTIFY
18	NOTIFY	**TimerExpired (NotifyTimeout)** => StartTimer (NotifyTimeout) AddFlags.Notification := 1 R_SendNotify_req	NOTIFY
19	NOTIFY	**R_SendNotify_cnf** => ignore	NOTIFY

#	Current State	Event /Condition =>Action	Next State
20	NOTIFY	**LMPM_R_Data.ind** /CheckRSI() == RespondError => R_SendError_req	NOTIFY
21	NOTIFY	**LMPM_R_Data.ind** /CheckRSI() != RespondError => ignore	NOTIFY
22	NOTIFY	**R_SendError_cnf** => ignore	NOTIFY

4.8.4.7.5 Functions, Macros, Timers and Variables

Table 302 contains the functions, macros, timers and variables used by the RSIRN, their arguments and their descriptions.

Table 302 – Functions, Macros, Timers and Variables used by the RSIRN

Name	Type	Function/Meaning
NotifyTimeout	Timer	Timeout for Notification repetition. The value of RSII::FragTimeout is used (2 seconds)
StartTimer	Function	This local function is used to start or restart a timer.
StopTimer	Function	This local function is used to stop a timer.
TimerExpired	Function	This local function is issued if a timer expires.
RSIR	Variable	Associated state machine
R_SendNotify_req	Macro	/* Send a RTA-ACK-PDU */ PDU.DA := IMAC PDU.DestinationSAP := ISAP PDU.SourceSAP := RSAP PDU.PDUType.Version := 2 PDU.PDUType.Type := ACK PDU.AddFlags.WindowSize := WindowSize PDU.AddFlags.TACK := 0 PDU.AddFlags.MoreFrag := 0 PDU.AddFlags.Notification := (RSIRN.state == NOTIFY) ? 1 : 0 PDU.SendSeqNum := SendSeqNum PDU.AckSeqNum := RecvSeqNum PDU.VarPartLen := 0 LMPM_R_Data.req
R_SendNotify_cnf	Macro	LMPM_R_Data.cnf /PDU.PDUType.Type == ACK
CheckRSI()	Macro	See RSIR::CheckRSI()
R_SendError_req	Macro	See RSIR::R_SendError_req
R_SendError_cnf	Macro	See RSIR::R_SendError_cnf

4.8.5 DLL Mapping Protocol Machines

There is no DLL Mapping Protocol Machine (DMPM) defined for this Protocol.

4.9 Fragmentation

4.9.1 General

The LengthOfPeriod is limited at the lower end by the maximum size of an Ethernet frame. The fragmentation reduces, using a peer to peer approach, the conveyed size of an Ethernet frame and allows to use a LengthOfPeriod down to 31,25 µs.

The fragmentation uses the IEEE Std 802.1Q defined TCI.PCP to encode and decode up to 8 parallel fragmentation streams (at least one shall be supported). Because of the bundling to the IEEE Std 802.1Q the granularity of the supported streams shall be one, two, four, or eight and the assignment to the TCI.PCP is shown in Table 303.

Table 303 – TCI.PCP vs. streams

<table>
<tr><th>Priority</th><th>One stream</th><th>Two streams</th><th>Four streams</th><th>Eight streams</th></tr>
<tr><td>7</td><td rowspan="8">Default
All priorities are assigned to one stream</td><td rowspan="4">The upper four priorities are assigned to one stream</td><td rowspan="2">Two priorities are assigned to one stream</td><td>One priority is one stream</td></tr>
<tr><td>6</td><td>One priority is one stream</td></tr>
<tr><td>5</td><td rowspan="2">Two priorities are assigned to one stream</td><td>One priority is one stream</td></tr>
<tr><td>4</td><td>One priority is one stream</td></tr>
<tr><td>3</td><td rowspan="4">The lower four priorities are assigned to one stream</td><td rowspan="2">Two priorities are assigned to one stream</td><td>One priority is one stream</td></tr>
<tr><td>2</td><td>One priority is one stream</td></tr>
<tr><td>1</td><td rowspan="2">Two priorities are assigned to one stream</td><td>One priority is one stream</td></tr>
<tr><td>0</td><td>One priority is one stream</td></tr>
</table>

Even for the frames generated by the fragmentation the lower frame size limit of the IEEE Std 802.3 applies (64 octets without VLAN and 68 octets with VLAN). Thus the use of the principle of fragmentation is limited according to Figure N.2, Formula (39), Formula (40), Formula (41) and Formula (42) as shown in Table 304.

The following Formula (39) shows the calculation for a single frame which is fragmented into a first and a last fragment.

NOTE The LowestFractionizeableFrameSize has two different values for frames with or without VLAN. Thus the fragmentation payload is defined and used as a simplification.

$$\text{LowestFractionizeableFrameSize} = \text{FrameDataFromFirstFragment} + \text{FrameDataFromLastFragment} + \text{ProtocolOverhead} \quad (39)$$

With VLAN

LowestFractionizeableFrameSize = 40 octets + 38 octets + 20 octets

LowestFractionizeableFrameSize = 98 octets

Without VLAN

LowestFractionizeableFrameSize = 48 octets + 42 octets + 16 octets

LowestFractionizeableFrameSize = 106 octets

where

LowestFractionizeableFrameSize	is the minimum length of frame to be able to be fractionized
FrameDataFromFirstFragment	is the amount of data of the original frame which is conveyed by the first fragment
FrameDataFromLastFragment	is the amount of data of the original frame which is conveyed by the last fragment
ProtocolOverhead	is the amount of data of the Ethernet protocol frame

The following Formula (40) shows the calculation for the minimum residue payload of a frame which is fragmented into a middle and a last fragment.

NOTE The LowestFractionizeableFrameSize has two different values for frames with or without VLAN. For simplification only the value with VLAN is used.

LowestFractionizeableFrameSize = FrameDataFromMiddleFragment + FrameDataFromLastFragment + ProtocolOverhead (40)

With VLAN

LowestFractionizeableFrameSize = 40 octets + 38 octets + 26 octets

LowestFractionizeableFrameSize = 104 octets

Without VLAN

LowestFractionizeableFrameSize = 48 octets + 42 octets + 22 octets

LowestFractionizeableFrameSize = 112 octets

where

LowestFractionizeableFrameSize	is the minimum length of frame to be able to be fractionized
FrameDataFromMiddleFragment	is the amount of residual data of the original frame which is conveyed by a middle fragment
FrameDataFromLastFragment	is the amount of residual data of the original frame which is conveyed by the last fragment
ProtocolOverhead	is the amount of data of the Ethernet protocol frame

Formula (41) shows a frame used for the first, middle or last fragment with a VLAN tag.

Lowest frame size(with VLAN) = DestinationMAC + SourceMAC + VLAN + FragmentationHeader + Payload + FCS (41)

Payload(with VLAN) = Lowest frame size – (DestinationMAC + SourceMAC + VLAN + FragmentationHeader + FCS)

Payload(with VLAN) = 64 – (6 + 6 + 4 + 6 + 4)

Payload(with VLAN) = 38

according to the rules for FragStatus.MoreFollows:=More fragments follow

Payload(with VLAN) = 40

according to the rules for FragStatus.MoreFollows:=Last fragment

Payload(with VLAN) = 38

where

Lowest frame size	is the allowed minimum length of a frame (64 octets)
DestinationMAC	is the length of the MAC address (6 octets)
SourceMAC	is the length of the MAC address (6 octets)
VLAN	is the length of the VLAN tag (4 octets)
FragmentationHeader	is the length of the fragmentation header (6 octets)
Payload	is the length of the payload
FCS	is the length of the FCS (4 octets)

Formula (42) shows a frame used for the first, middle or last fragment without a VLAN tag.

Lowest frame size(without VLAN) = DestinationMAC + SourceMAC + FragmentationHeader + Payload + FCS (42)

Payload(without VLAN) = Lowest frame size – (DestinationMAC + SourceMAC + FragmentationHeader + FCS)

Payload(without VLAN) = 64 – (6 + 6 + 6 + 4)

Payload(without VLAN) = 42

according to the rules for FragStatus.MoreFollows:=More fragments follow

Payload(without VLAN) = 48

according to the rules for FragStatus.MoreFollows:=Last fragment

Payload(without VLAN) = 42

where

Lowest frame size	is the allowed minimum length of a frame (64 octets)
DestinationMAC	is the length of the MAC address (6 octets)
SourceMAC	is the length of the MAC address (6 octets)
VLAN	is the length of the VLAN tag (4 octets)
FragmentationHeader	is the length of the fragmentation header (6 octets)
Payload	is the length of the payload
FCS	is the length of the FCS (4 octets)

Table 304 – Lower limit of fragments

Case	VLAN	Payload size of the fragments	Meaning
First + last	With	Payload first fragment := 40 octets Payload last fragment := 38 octets Protocol part := 20 octets	Smallest fragment able frame size := DA + SA + VLAN + Payload + FCS := 98 octets
	Without	Payload first fragment := 48 octets Payload last fragment := 42 octets Protocol part := 16 octets	Smallest fragment able frame size := DA + SA + Payload + FCS := 106 octets
Middle + last	With	Payload middle fragment := 40 octets Payload last fragment := 38 octets Protocol part := 26 octets	Smallest fragment able residual payload size := DA + SA + VLAN + FragHeader + Payload + FCS := 104 octets
	Without	Payload middle fragment := 48 octets Payload last fragment := 42 octets Protocol part := 22 octets	Smallest fragment able residual payload size := DA + SA + FragHeader + Payload + FCS := 112 octets

A fragmentation payload of 90 octets for all kinds of frames (with and without VLAN) may be used to simplify the implementation of fragmentation. Thus the lowest frame size which could still be fragmented shall be 110 octets with VLAN and 106 octets without VLAN and the lowest residual fragmentation payload which could still be fragmented shall be 116 octets with VLAN and 112 octets without VLAN.

4.9.2 FRAG syntax description

4.9.2.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.9.2.2 FRAG APDU abstract syntax

Table 305 defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 305 shall represent the content of the DLSDU in Table 10.

Table 305 – FRAG APDU syntax

APDU name	APDU structure
FRAG-PDU	FRAG-DATA-PDU

Table 306 defines structures for substitutions of elements of the APDU structures shown in Table 305.

Table 306 – FRAG substitutions

Substitution name	Structure
FRAG-DATA-PDU	FragDataLength, FragStatus, FragData

4.9.3 FRAG transfer syntax

4.9.3.1 Coding section related to FRAG-PDUs specific fields

4.9.3.1.1 Coding of the field FragDataLength

The field FragDataLength is coded as an Unsigned8 and contains the number of 8 octet portions of the fragment data. For the last fragment (FragStatus.MoreFollows == 0) up to 7 “residue” octets may be added.

FragDataLength shall be calculated according to Formula (43).

$$\text{FragDataLength} = (\text{Length of FragData } DIV\ 8) \quad (43)$$

where

FragDataLength	is the length of the field FragData in 8 octet portions
Length of FragData	is the length of the field FragData in octets
8	is the defined value of an increment of FragData

If the fragmentation of a frame into multiples of 8 octet portions has any residues, then the size of the last fragment containing the FragData shall be the rest, containing the residue octets.

The receiver shall accept a fragmented frame with this kind of FragData if FragStatus.MoreFollows == 0.

This field shall be coded as Unsigned8 with the values according to Table 307.

Table 307 – FragDataLength

Value (hexadecimal)	Meaning	Usage
0x00, …, 0x04	—	Reserved
0x05 or 0x06 [a]	Minimum If the FragStatus.MoreFollows := 1, then the length of FragData is 0x30 octets If the FragStatus.MoreFollows := 0, then the length of FragData is 0x30 to 0x37 octets (up to 7 residue octets)	Mandatory
0x07, …, 0xBE	…	Mandatory
0xBF [b]	Maximum If the FragStatus.MoreFollows := 1, then the length of FragData is 0x5CF octets If the FragStatus.MoreFollows := 0, then the length of FragData is 0x5F8 to 0x5FF octets	Mandatory
0xC0, …, 0xFF	Maximum length of FragData is 0x7FF	Optional

[a] The minimum FragDataLength which shall be supported by the DEFRAG is 0x05 with VLAN and 0x06 without VLAN.

[b] The maximum FragDataLength which shall be supported by the DEFRAG is defined by the use of at least two fragments for a maximum IEEE Std 802.3 frame. The support of one-fragment fragmentation is not intended.

4.9.3.1.2 Coding of the field FragStatus

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0 – 5: FragStatus.FragmentNumber
This field contains the fragment number and shall be coded according to Table 308.

Table 308 – FragStatus.FragmentNumber

Value (hexadecimal)	Meaning
0x00	First fragment of a frame
0x01	Second fragment of a frame
…	…
0x3F	64th fragment of a frame

Bit 6: FragStatus.Reserved
This field shall be coded according to Table 309.

Table 309 – FragStatus.Reserved

Value (hexadecimal)	Meaning
0x00	Reserved

Bit 7: FragStatus.MoreFollows
This field shall be coded according to Table 310.

Table 310 – FragStatus.MoreFollows

Value (hexadecimal)	Meaning
0x00	Last fragment
0x01	More fragments follow

4.9.3.1.3 Coding of the field FragData

This field shall be coded as data type OctetString and shall contain the transmitted portion of data of the fragmented frame.

4.9.4 FAL Service Protocol Machines

There is no FAL Service Protocol Machine (FSPM) defined for this Protocol.

4.9.5 Application Relationship Protocol Machines

There is no Application Relationship Protocol Machine (ARPM) defined for this Protocol.

4.9.6 DLL Mapping Protocol Machines

4.9.6.1 Fragmentation

4.9.6.1.1 Fragmentation dynamic

4.9.6.1.1.1 Primitive definitions

4.9.6.1.1.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by FRAG_D are described in the service definition and shown in Table 311.

Table 311 – Remote primitives issued or received by FRAG_D

Primitive	Source	Destination	Associated parameters	Functions
QueueHandler.req	FRAG_D	QueueHandler	Queue, Frame	—

4.9.6.1.1.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by FRAG_D are described in the service definition and shown in Table 312.

Table 312 – Local primitives issued or received by FRAG_D

Primitive	Source	Destination	Associated parameters	Functions
SCHEDULER_StartOfPeriod	SCHEDULER	FRAG	—	This local function is issued if a new communication cycle starts.

4.9.6.1.1.2 State transition diagram

The state transition diagram of the FRAG_D is shown in Figure 97.

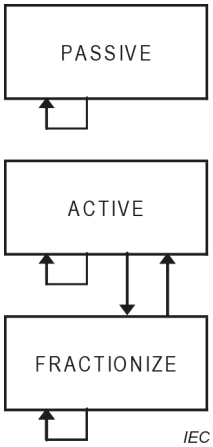


Figure 97 – State transition diagram of FRAG_D

States of the FRAG_D are:

PASSIVE	The FRAG unit is not used. Frames pass unaffected.
ACTIVE	The FRAG unit for the defined stream is active and waits for frames to fractionize them if needed.
FRACTIONIZE	A frame is fragmented and conveyed. High priority frames which do not need fragmentation may overtake the fragments. After the last fragment is sent, the state ACTIVE is reached. If residential time is available, the fragmentation starts immediately with the next frame.

4.9.6.1.1.3 State machine description

The FRAG is responsible for fragmentation of frames according to the residual GREEN period. It gets the frames from the QueueHandler, breaks them into fragments if necessary and allowed by the protocol, and stores the fragments in the lower queues of the QueueHandler.

In the inactive state FRAG shall be bypassed by the QueueHandler.

The fragmentation may be done by two different strategies. The first one is the dynamic fragmentation, which uses the residential GREEN phase and the second one is the static fragmentation, which uses a predefined fragment size.

4.9.6.1.1.4 FRAG_D state table (dynamic fragmentation)

Table 313 contains the FRAG_D state table.

Table 313 – FRAG_D state table (dynamic)

#	Current State	Event /Condition =>Action	Next State
1	PASSIVE	**FragmentationMode == No** => ignore	PASSIVE
2	PASSIVE	**FragmentationMode == Enabled** => ignore	ACTIVE
3	ACTIVE	**QueuesNotEmpty ()** /Period == RED => ignore	ACTIVE
4	ACTIVE	**QueuesNotEmpty ()** /Period == GREEN => GetFrameFromQueue () StartFragStream ()	FRACTIONIZE
5	FRACTIONIZE	**StartFragStream ()** /Period == GREEN && FrameFitsIntoResidualPeriod == TRUE => QueueHandler.req () //NOTE Send frame	ACTIVE
6	FRACTIONIZE	**StartFragStream ()** /Period == GREEN && FrameFitsIntoResidualPeriod == FALSE && FrameSizeAllowsFragmentation == TRUE && ResidualPeriodShorterThanMin == TRUE => ignore //NOTE Wait until the next GREEN period starts	FRACTIONIZE
7	FRACTIONIZE	**SCHEDULER_StartOfPeriod ()** /Period == GREEN && ResidualPeriodShorterThanMin == TRUE => ignore	FRACTIONIZE
8	FRACTIONIZE	**StartFragStream ()** /Period == GREEN && FrameFitsIntoResidualPeriod == FALSE && FrameSizeAllowsFragmentation == TRUE && ResidualPeriodShorterThanMin == FALSE && LastFragment == FALSE => CalculateFragSize () CreateFrag () QueueHandler.req () //NOTE Send frame fragment	FRACTIONIZE

#	Current State	Event /Condition =>Action	Next State
9	FRACTIONIZE	**SCHEDULER_StartOfPeriod ()** /Period == GREEN && ResidualPeriodShorterThanMin == FALSE && LastFragment == TRUE => CalculateFragSize () CreateFrag () QueueHandler.req () //NOTE Send frame fragment	ACTIVE
10	FRACTIONIZE	**QueuesNotEmpty ()** /CheckForFrameWithHigherPriority () == EXISTS && FrameFitsIntoResidualPeriod == TRUE => GetFrameFromQueue () QueueHandler.req ()	FRACTIONIZE
11	FRACTIONIZE	**QueuesNotEmpty ()** /CheckForFrameWithHigherPriority () == EXISTS && FrameFitsIntoResidualPeriod == FALSE => ignore	FRACTIONIZE

4.9.6.1.1.5 Functions, Macros, Timers and Variables

Table 314 contains the functions, macros, timers and variables used by the FRAG_D and their arguments and their descriptions.

Table 314 – Functions, Macros, Timers and Variables used by the FRAG_D (dynamic)

Name	Type	Function/Meaning
CalculateFragSize	Function	This local function calculates the fragment size due to the residual GREEN period and the fragmentation rules. If dynamic fragmentation is not supported, use a default length given by the engineering according to the minimal GREEN period of all phases.
CheckForFrameWithHigherPriority	Function	This local function tests whether any frame with a higher priority than the fragmented one is stored in the queues.
CreateFrag	Function	This local function creates a fragment according to the coding and the fragmentation rules. Either first, next or last. Keep in mind, that the minimum and the maximum frame size of a IEEE Std 802.3 frame are still valid, even for fragments.
FrameFitsIntoResidualPeriod	Function	This local function checks whether the residual GREEN period is longer than the time the frame needs for conveyance
FrameSizeAllowsFragmentation	Function	This local function checks whether the frame is big enough for fragmentation.
GetFrameFromQueue	Function	This local function fetches the next frame according to the amount of FRAG streams and priorities.
QueuesNotEmpty	Function	This local function is issued if any entry is stored in the queues.
ResidualPeriodShorterThanMin	Function	This local function checks whether the residual GREEN period is shorter than the time a minimum frame needs for conveyance

Name	Type	Function/Meaning
StartFragStream	Function	This local function triggers the fragment creation of FRAG.
LastFragment	Macro	This local macro contains information whether the actual fragment is the last fragment. With the conveyance of the last fragment the FRAG stream ends.
Period	Macro	This local macro contains the information which period (RED or GREEN) is active.

4.9.6.1.2 Fragmentation static

4.9.6.1.2.1 Primitive definitions

4.9.6.1.2.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by FRAG_S are described in the service definition and shown in Table 315.

Table 315 – Remote primitives issued or received by FRAG_S

Primitive	Source	Destination	Associated parameters	Functions
QueueHandler.req	FRAG_S	QueueHandler	Queue, Frame	—

4.9.6.1.2.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by FRAG_S are described in the service definition and shown in Table 316.

Table 316 – Local primitives issued or received by FRAG_S

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.9.6.1.2.2 State transition diagram

The state transition diagram of the FRAG_S is shown in Figure 98.

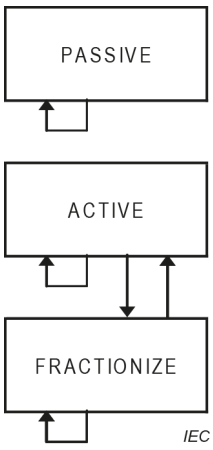


Figure 98 – State transition diagram of FRAG_S

States of the FRAG_S are:

PASSIVE	The FRAG unit is not used. Frames pass unaffected.
ACTIVE	The FRAG unit for the defined stream is active and waits for frames to fractionize them if needed.
FRACTIONIZE	A frame is fragmented and conveyed. High priority frames which do not need fragmentation may overtake the fragments. After the last fragment is sent, the state ACTIVE is reached. If residential time is available, the fragmentation starts immediately with the next frame.

4.9.6.1.2.3 State machine description

The FRAG is responsible for fragmentation of frames according to the residual GREEN period. It gets the frames from the QueueHandler, breaks them into fragments if necessary and allowed by the protocol, and stores the fragments in the lower queues of the QueueHandler.

In the inactive state FRAG shall be bypassed by the QueueHandler.

4.9.6.1.2.4 FRAG_S state table (static fragmentation)

Table 317 contains the FRAG_S state table.

Table 317 – FRAG_S state table (static)

#	Current State	Event /Condition =>Action	Next State
1	PASSIVE	**FragmentationMode == No** => ignore	PASSIVE
2	PASSIVE	**FragmentationMode == Enabled** && RTC3PSM.PortState != OFF => ignore	ACTIVE
3	ACTIVE	**FragmentationMode == Enabled** && RTC3PSM.PortState == OFF => ignore	PASSIVE
4	ACTIVE	**QueuesNotEmpty ()** /Period == RED => ignore	ACTIVE

5	ACTIVE	**QueuesNotEmpty ()**
		/Period == GREEN
		&&
		FrameSizeAllowsFragmentation == FALSE
		=>
		GetFrameFromQueue ()
		
		// Adapt the assigned queues if more than one Frag stream exists
		QueueHandler.req ()
		
		//NOTE Send frame
			ACTIVE

6	ACTIVE	**QueuesNotEmpty ()** /Period == GREEN && FrameSizeAllowsFragmentation == TRUE => GetFrameFromQueue () // Adapt the assigned queues if more than one Frag stream exists StartFragStream ()	FRACTIONIZE

#	Current State	Event /Condition =>Action	Next State
7	FRACTIONIZE	**StartFragStream ()** => CalculateFragSize () CreateAllFrags () For all fragment frames QueueHandler.req () //NOTE Send fragment frame	FRACTIONIZE
8	FRACTIONIZE	**LastFragmentTransmitted ()** => ignore	ACTIVE

9	FRACTIONIZE	**QueuesNotEmpty ()**
		/CheckForFrameWithHigherPriority () == EXISTS
		&&
		FrameSizeAllowsFragmentation == FALSE
		=>
		GetFrameFromQueue ()
		
		// Adapt the assigned queues if more than one Frag stream exists
		QueueHandler.req ()
		
		//NOTE Send frame
			FRACTIONIZE

10	FRACTIONIZE	**QueuesNotEmpty ()** /CheckForFrameWithHigherPriority () == EXISTS && FrameSizeAllowsFragmentation == TRUE => ignore	FRACTIONIZE

4.9.6.1.2.5 Functions, Macros, Timers and Variables

Table 318 contains the functions, macros, timers and variables used by the FRAG_S and their arguments and their descriptions.

Table 318 – Functions, Macros, Timers and Variables used by the FRAG_S (static)

Name	Type	Function/Meaning
CalculateFragSize	Function	This local function calculates the fragment size due to the given expected fragment size and the fragmentation rules. The default length is given by the engineering according to the minimal GREEN period of all phases.
CheckForFrameWithHigherPriority	Function	This local function tests whether any frame with a higher priority than the fragmented one is stored in the queues.
CreateAllFrags	Function	This local function creates all fragments according to the coding and the fragmentation rules. Either first, next or last. Keep in mind, that the minimum and the maximum frame size of a IEEE Std 802.3 frame are still valid, even for fragments.
FrameSizeAllowsFragmentation	Function	This local function checks whether the frame is big enough for fragmentation
GetFrameFromQueue	Function	This local function fetches the next frame according to the amount of FRAG streams and priorities.
QueuesNotEmpty	Function	This local function is issued if any entry is stored in the queues.
StartFragStream	Function	This local function triggers the fragment creation of FRAG.
LastFragmentTransmitted	Function	This local function is issued when the last fragment is conveyed.

Name	Type	Function/Meaning
Period	Macro	This local macro contains the information which period (RED or GREEN) is active.

4.9.6.2 Defragmentation

4.9.6.2.1 Primitive definitions

4.9.6.2.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DEFRAG are described in the service definition and shown in Table 319.

Table 319 – Remote primitives issued or received by DEFRAG

Primitive	Source	Destination	Associated parameters	Functions
DMUX_DEFRAG.ind	DEMUX	DEFRAG	Port, Tstamp, DA, SA, VLAN, N_SDU	—
DMUX_MAC_RELAY.ind	DEFRAG	MAC_RELAY	Port, Tstamp, DA, SA, N_SDU	—

4.9.6.2.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DEFRAG are described in the service definition and shown in Table 320.

Table 320 – Local primitives issued or received by DEFRAG

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.9.6.2.2 State transition diagram

The state transition diagram of the DEFRAG is shown in Figure 99.

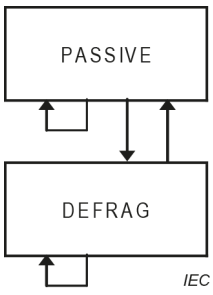


Figure 99 – State transition diagram of DEFRAG

States of the DEFRAG are:

PASSIVE	Waiting for the first fragment of a stream. For each supported stream there exists a DEFRAG unit.
DEFRAG	A stream was started by the first fragment. After receiving the last fragment of the current stream, the machine reaches PASSIVE again. If the amount of data received during DEFRAG exceeds the maximum frame size according to the IEEE Std 802.3, the defragmentation is aborted and the state PASSIVE is entered.

4.9.6.2.3 State machine description

A DEFRAG state machine shall exist for each supported stream. At least one stream shall be supported, if fragmentation is supported. In this case the DEFRAG is waiting for a FRAGMENT after the link has been active.

4.9.6.2.4 DEFRAG state table

Table 321 contains the DEFRAG state table which exists for each port.

Table 321 – DEFRAG state table

#	Current State	Event /Condition =>Action	Next State
1	PASSIVE	**DMUX_DEFRAG.ind ()** /DefragGuard () == FirstFragment => Store fragment	DEFRAG
2	PASSIVE	**DMUX_DEFRAG.ind ()** /DefragGuard () == Invalid => ignore	PASSIVE
3	DEFRAG	**DMUX_DEFRAG.ind ()** /DefragGuard () == NextFragment && FrameLimitReached => Delete stored fragments	PASSIVE
4	DEFRAG	**DMUX_DEFRAG.ind ()** /DefragGuard () == NextFragment && ! FrameLimitReached => Store fragment	DEFRAG
5	DEFRAG	**DMUX_DEFRAG.ind ()** /DefragGuard () == LastFragment && FrameLimitReached => Delete stored fragments	PASSIVE
6	DEFRAG	**DMUX_DEFRAG.ind ()** /DefragGuard () == LastFragment && ! FrameLimitReached => Create frame Delete stored fragments DMUX_MAC_RELAY.ind ()	PASSIVE
7	DEFRAG	**DMUX_DEFRAG.ind ()** /DefragGuard () == FirstFragment => Delete stored fragments Store fragment	DEFRAG

#	Current State	Event /Condition =>Action	Next State
8	DEFRAG	**DMUX_DEFRAG.ind ()** /DefragGuard () == Invalid => Delete stored fragments	PASSIVE

4.9.6.2.5 Functions, Macros, Timers and Variables

Table 322 contains the functions and macros used by the DEFRAG, their arguments and their descriptions.

Table 322 – Functions, Macros, Timers and Variables used by the DEFRAG

Name	Type	Function/Meaning
DefragGuard	Function	This local function checks the fragments against truth tables.
Store fragment	Macro	This local macro stores the fragment for further defragmentation.
Delete stored fragment	Macro	This local macro deletes the previous stored fragments.
Create frame	Macro	This local macro creates a frame from the stored fragments.
FrameLimitReached	Macro	This local macro checks whether the IEEE Std 802.3 frame limits are exceeded.

4.9.6.2.6 DefragGuard

The DefragGuard checks the received frame against Table 323, Table 324 and Table 325. The used Table depends on the locally supported functionality.

The defragmentation shall accept a frame independent of the value of FrameID.FragSequence in combination with FragStatus.FragmentNumber == 0 as start of a new defragmentation sequence.

Table 323 – Truth table for the DefragGuard – first fragment

Input				Output
FragmentationFrameID. FragSequence	FragStatus. FragmentNumber	FragStatus. MoreFollows	FragDataLength	Local
Invalid	—	—	—	Invalid
Valid	Invalid	—	—	Invalid
Valid	Valid	FALSE [a]	—	Invalid
Valid	Valid	TRUE	Invalid	Invalid
Valid	Valid	TRUE	Valid	Valid

[a] The MoreFollows for the first fragment shall never be set to FALSE by the sender.

Table 324 – Truth table for the DefragGuard – next fragment

Input				Output
FragmentationFrameID. FragSequence	**FragStatus. FragmentNumber**	**FragStatus. MoreFollows**	**FragDataLength**	**Local**
Invalid	—	—	—	Invalid
Valid [a]	Invalid	—	—	Invalid
Valid	Valid [b]	FALSE	—	Invalid
Valid	Valid [b]	TRUE	Invalid	Invalid
Valid	Valid	TRUE	Valid	Valid

[a] Identical to the stored one of the first fragment

[b] The stored one from the last received fragment of this FragSequence (FragmentNumber == 1)

Table 325 – Truth table for the DefragGuard – last fragment

Input				Output
FragmentationFrameID. FragSequence	**FragStatus. FragmentNumber**	**FragStatus. MoreFollows**	**FragDataLength**	**Local**
Invalid	—	—	—	Invalid
Valid	Invalid	—	—	Invalid
Valid	Valid	TRUE	—	Invalid
Valid	Valid	FALSE [a]	Invalid	Invalid
Valid	Valid	FALSE [a]	Valid	Valid

[a] The MoreFollows for the last fragment shall be set to FALSE by the sender.

4.10 Remote procedure call

4.10.1 General

For more detailed info about the usage of the fields defined in 4.10 see “Publication C706 of The Open Group” or “The Open Group, Technical Standard, DCE 1.1 – Remote Procedure Call”.

4.10.2 RPC syntax description

4.10.2.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.10.2.2 RPC APDU abstract syntax

Table 326 shows the utilization of the Publication C706 of The Open Group and defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 326 shall represent the content of the DLSDU in Table 10.

Table 326 – RPC APDU syntax

APDU name	APDU structure
CL-RPC-PDU	IPHeader, UDPHeader, RPC-SDU
RPC-SDU	RPCHeader, NDRData
NDRData	NDRDataRequest ^ NDRDataResponse ^ NDREPMapLookupReq ^ NDREPMapLookupRes ^ NDREPMapLookupFreeReq ^ NDREPMapLookupFreeRes ^ NDRAck ^ NDRQuAck ^ NDRQuit ^ NDRFack ^ NDRFault ^ NDRNoCall ^ NDRWorking ^ NDRPing ^ NDRReject

Table 327 defines structures for substitutions of elements of the APDU structures shown in Table 326.

Table 327 – RPC substitutions

Substitution name	Structure
RPCHeader	RPCVersion (4), RPCPacketType, RPCFlags, RPCFlags2, RPCDRep, RPCSerialHigh, RPCObjectUUID [a], RPCInterfaceUUID [b], RPCActivityUUID, RPCServerBootTime, RPCInterfaceVersion, RPCSequenceNmb, RPCOperationNmb [c], RPCInterfaceHint, RPCActivityHint, RPCLengthOfBody, RPCFragmentNmb, RPCAuthenticationProtocol, RPCSerialLow
NDRDataRequest	ArgsMaximum, ArgsLength, MaximumCount, Offset(0), ActualCount, PROFINETIOServiceReqPDU
NDRDataResponse	PNIOStatus, ArgsLength, MaximumCount, Offset(0), ActualCount, [PROFINETIOServiceResPDU]
NDRFault	RPCNCAFaultStatus
NDRAck	NULL
NDRQuAck	NULL ^ (RPCCancelVersion(0), RPCCancelID, RPCServerIsAccepting)
NDRQuit	NULL ^ (RPCCancelVersion(0), RPCCancelID)
NDRFack	NULL ^ (RPCVersionFack, RPCPad1, RPCWindowSize, RPCMaxTsdu, RPCMaxFragSize, RPCSerialNumber, RPCSelAckLen, RPCArrayOfSelAck*)
NDRNoCall	NULL ^ NDRFack
NDRWorking	NULL
NDRPing	NULL
NDRReject	RPCNCARejectStatus
NDREPMapLookupReq	RPCInquiryType, RPCObjectReference(1), RPCObjectUUID, RPCInterfaceReference(2), RPCInterfaceUUID, RPCInterfaceVersionMajor, RPCInterfaceVersionMinor, RPCVersionOption(1), RPCEntryHandleAttribute(0), RPCEntryHandleUUID, RPCMaxEntries
NDREPMapLookupRes	RPCEntryHandleAttribute(0), RPCEntryHandleUUID, RPCNumberOfEntries, RPCMaxEntries, RPCEntriesOffset, RPCEntriesCount, [(RPCObjectUUID, RPCTowerReference, RPCAnnotationOffset, RPCAnnotationLength, RPCAnnotation, [RPCGap*], RPCTowerLength, RPCTowerOctetStringLength, [RPCTowerOctetString*], [RPCGap*])*], RPCEPMapStatus
RPCTowerOctetString	RPCFloorCount, [RPCFloor*]
RPCFloor	RPCLHSByteCount, [RPCProtocolIdentifierString*], RPCRHSByteCount, [RPCRelatedData*]
RPCProtocolIdentifier-String	(RPCID, RPCInterfaceUUID [d], RPCInterfaceVersionMajor [d]) ^ (RPCID, RPCDataRepresentationUUID [d], RPCInterfaceVersionMajor [d]) ^ RPCProtocolIdentifier ^ RPCServerUDPPort ^ RPCHostAddress
RPCRelatedData	RPCInterfaceVersionMinor [d] ^ RPCPortNumber ^ RPCIPAddress
NDREPMapLookupFree Req	RPCEntryHandleAttribute, RPCEntryHandleUUID

Substitution name	Structure
NDREPMapLookupFree Res	RPCEntryHandleAttribute, RPCEntryHandleUUID, RPCEPMapStatus
RPCAnnotation	DeviceType, Blank, OrderID, Blank, HWRevision, Blank, SWRevisionPrefix, SWRevision, EndTerm

[a] To identify IO device, IO controller, IO supervisor

[b] To identify PNIO interface type

[c] To identify the service type, for example Connect, Release, Read, Write, Control

[d] Shall be coded in little endian format

4.10.3 RPC Transfer syntax

4.10.3.1 General

This document uses idempotent RPC services which are executed “at-least-once”. “At-least-once” executes the requested RPC service each time it is received. Retries are not detected as duplicates.

4.10.3.2 Coding section related to CL-RPC-PDU

4.10.3.2.1 Coding of the field RPCVersion

This field shall be coded as data type Unsigned8 with values according to Table 328.

Table 328 – RPCVersion

Value (hexadecimal)	Meaning
0x04	Used RPC version
Other	Reserved

4.10.3.2.2 Coding of the field RPCPacketType

This field shall be coded as data type Unsigned8 with values according to Table 329.

Table 329 – RPCPacketType

Value (hexadecimal)	Meaning
0x00	Request
0x01	Ping
0x02	Response
0x03	Fault
0x04	Working
0x05	No call, response to ping
0x06	Reject
0x07	Acknowledge
0x08	Connectionless cancel
0x09	Fragment acknowledge (FACK-PDU)
0x0A	Cancel acknowledge
0x0B – 0xFF	Reserved

The decoder shall only check the 5 least significant bits.

4.10.3.2.3 Coding of the field RPCFlags

This field shall be coded as data type according to 3.4.2.3.3 with values according to Table 330.

Table 330 – RPCFlags

Bit	Meaning if set to 1
Bit 0	Implementation specific, shall be set to zero
Bit 1	Last fragment
Bit 2	Fragment
Bit 3	No fragment acknowledge requested
Bit 4	Maybe
Bit 5	Idempotent
Bit 6	Broadcast
Bit 7	Implementation specific, shall be set to zero

4.10.3.2.4 Coding of the field RPCFlags2

This field shall be coded as data type according to 3.4.2.3.3 with values according to Table 331.

Table 331 – RPCFlags2

Bit	Meaning if set to 1
Bit 0	Implementation specific, shall be set to zero
Bit 1	Cancel was pending at end of call
Bit 2	Reserved
Bit 3	Reserved
Bit 4	Reserved
Bit 5	Reserved
Bit 6	Reserved
Bit 7	Reserved

4.10.3.2.5 Coding of the field RPCDRep

This field shall be coded as data type OctetString[3].

RPCDRep Octet 1
The coding of the first octet shall be according to 3.4.2.3 and the individual bits shall have the meaning defined in Table 332.

Table 332 – RPCDRep.Character- and IntegerEncoding

Bit	Field Name	Value	Meaning
0 – 3	RPCDRep.CharacterEncoding [a]	0	ASCII
		1	EBCDIC
4 – 7	RPCDRep.IntegerEncoding [b]	0	Big endian [c]
		1	Little endian [c]

[a] Not used within Type 10.

[b] As an exception to 3.4.2.3.7 the RPC encoding rules are applied to the RPCHeader, NDR substitutions and NDREPMap substitutions within Type 10. It only uses Unsigned8, Unsigned16, or Unsigned32 there. The user data of the Type 10 service definitions are not affected because everything is an opaque OctetString there.

[c] See The Open Group – Publication C706.

RPCDRep Octet 2
The values of the second octet shall be encoded according to Table 333.

Table 333 – RPCDRep Octet 2 – Floating Point Representation

Value (hexadecimal)	Meaning
0x00	IEEE
0x01	VAX
0x02	CRAY
0x03	IBM
0x04 – 0xFF	Reserved

RPCDRep Octet 3
The value of the third octet shall be zero.

4.10.3.2.6 Coding of the field RPCSerialHigh

This field shall be coded as data type Unsigned8. The value contains the high octet of the fragment number of the call.

4.10.3.2.7 Coding of the field RPCSerialLow

This field shall be coded as data type Unsigned8. The value contains the low octet of the fragment number of the call.

NOTE The value of the field RPCSerial is incremented which each transmission, even with a transmission repetition in distinction to the field FragmentNmb.

4.10.3.2.8 Coding of the field RPCObjectUUID

This field shall be coded as data type Structure containing the following elements:

– Data1 as Unsigned32
– Data2 as Unsigned16
– Data3 as Unsigned16
– Data4 as array of eight Unsigned8 (Octet 1 to Octet 8)

The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The ordering of octets of Data4 is shown in Table 334.

Table 334 – RPCObjectUUID.Data4

Octet	Meaning
1	Values, see Table 335
2	
3	Instance High
4	Instance Low
5	DeviceID High
6	DeviceID Low
7	VendorID High
8	VendorID Low

Defined values for this document are shown in Table 335. These values shall be used in conjunction with the values from Table 336.

Table 335 – RPCObjectUUID for devices

Value	Description	
UUID_IO_ObjectInstance_XYZ DEA00000-6C97-11D1-8271-{xxxxyyyyzzzz}	Identifies an object instance within a physical device in case there are more than one instances, where	
	xxxx	See Table 615 Represents the instance or node number
	yyyy	See Table 617 Identifies the Device ID as a vendor specific number for the device class
	zzzz	See Table 618 Represents the Vendor Identification (VendorID) as a central administrative number assigned by the responsible user organization

4.10.3.2.9 Coding of the field RPCInterfaceUUID

This field shall be coded as data type Structure containing the following elements:

– Data1 as Unsigned32
– Data2 as Unsigned16
– Data3 as Unsigned16
– Data4 as array of eight Unsigned8

The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian) with the exceptions defined in Table 327.

Defined values for this document are shown in Table 336.

Table 336 – RPCInterfaceUUID for PNIO

Value	Description
UUID_IO_DeviceInterface DEA00001-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO device uniquely. RPCInterfaceVersion shall be: Major version: 1 Minor version: 0
UUID_IO_ControllerInterface DEA00002-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO controller uniquely. RPCInterfaceVersion shall be: Major version: 1 Minor version: 0
UUID_IO_SupervisorInterface DEA00003-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO supervisor uniquely. RPCInterfaceVersion shall be: Major version: 1 Minor version: 0
UUID_IO_ParameterServerInterface DEA00004-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO parameter server uniquely. RPCInterfaceVersion shall be: Major version: 1 Minor version: 0
UUID_IO_CIMInterface DEA00005-6C97-11D1-8271-00A02442DF7D	Identifies the interface of a CIM uniquely. This interface is used by NME and Query Stream. RPCInterfaceVersion shall be: Major version: 1 Minor version: 0

Defined values for the RPC endpoint mapper are shown in Table 337.

Table 337 – RPCInterfaceUUID for the RPC endpoint mapper

Value	Description
UUID_EPMap_Interface E1AF8308-5D1F-11C9-91A4-08002B14A0FA UUID_EPMap_Object 00000000-0000-0000-0000-000000000000	Identifies the interface of the endpoint mapper. The RPCInterfaceVersion shall be: Major version: 3 Minor version: 0

4.10.3.2.10 Coding of the field RPCActivityUUID

This field shall be coded as data type Structure containing the following elements:

– Data1 as Unsigned32
– Data2 as Unsigned16
– Data3 as Unsigned16
– Data4 as array of eight Unsigned8

NOTE The response mirrors the content of the field of the request.

The endianess is according to the value in the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The RPC client of an IOC and an IOD generates an RPCActivityUUID for each AR and uses it as long as the AR exists.

Any RPC client continuously using Implicit Read with an IOD or IOC shall generate one RPCActivityUUID for this communication relation and use it as long as this communication relation exist.

4.10.3.2.11 Coding of the field RPCServerBootTime

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

NOTE Idempotent functions do not need to maintain this value. Only the endpoint mapper is allowed to answer with zero.

4.10.3.2.12 Coding of the field RPCInterfaceVersion

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 15: RPCInterfaceVersion.Major
This field shall be set according to Table 338.

Table 338 – RPCInterfaceVersion.Major

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0xFFFF	Valid version information

Bit 16 – 31: RPCInterfaceVersion.Minor
This field shall be set according to Table 339.

Table 339 – RPCInterfaceVersion.Minor

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Valid version information

The endianess is according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.2.13 Coding of the field RPCSequenceNmb

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

This field is used together with the RPCActivityUUID to uniquely identify an RPC call.

4.10.3.2.14 Coding of the field RPCOperationNmb

This field shall be coded as data type Unsigned16. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The RPCOperationNmb identifies the PNIO service supported by the PNIO interfaces according to Table 336. The values for PNIO services are defined in Table 340.

Table 340 – RPCOperationNmb

Value (decimal)	Service	Usage
0	Connect	—
1	Release	—
2	Read	Only valid with ARUUID<>0
3	Write	Only valid with ARUUID<>0
4	Control	—
5	Read Implicit	Only valid with ARUUID=0
6 – 9	Reserved [a]	—
10	SecurityAssociationControl	Reserved for security. Establishment of a secured communication channel based on RPC or RSI
11 – 65 535	Reserved	—
[a] These values are used by RSI and reserved for RPC. See 4.8.2.2.1.		

The values for endpoint mapper services according to Table 341 shall be used.

Table 341 – RPCOperationNmb for endpoint mapper

Value (decimal)	Usage	Service
0	Optional	Insert
1	Optional	Delete
2	Mandatory	Lookup
3	Optional	Map
4	Mandatory	LookupHandleFree
5	Optional	InqObject
6	Optional	MgmtDelete
7 – 65 535	Reserved	—

4.10.3.2.15 Coding of the field RPCInterfaceHint

This field shall be coded as data type Unsigned16. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value should be set to no hint (0xFFFF) for this version.

The client shall start with “no hint” for the first call and should use the server response for the following calls to allow a server optimization.

4.10.3.2.16 Coding of the field RPCActivityHint

This field shall be coded as data type Unsigned16. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value should be set to no hint (0xFFFF) for this version.

The client shall start with “no hint” for the first call and should use the server response for the following calls to allow a server optimization.

4.10.3.2.17 Coding of the field RPCLengthOfBody

This field shall be coded as data type Unsigned16. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to the number of octets of NDRData of the current frame.

The conveyance of NDRData may require more than one frame. In this case RPCLengthOfBody contains only the number of octets for the current frame.

4.10.3.2.18 Coding of the field RPCFragmentNmb

This field shall be coded as data type Unsigned16. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to the number of the current fragment.

4.10.3.2.19 Coding of the field RPCAuthenticationProtocol

This field shall be coded as data type Unsigned8.

The value shall be set to zero for no authentication.

4.10.3.2.20 Coding of the field RPCCancelVersion

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to zero.

4.10.3.2.21 Coding of the field RPCCancelID

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.2.22 Coding of the field RPCServerIsAccepting

This field shall be coded as data type Unsigned8.

4.10.3.2.23 Coding of the field RPCVersionFack

This field shall be coded as data type Unsigned8 according to Table 342.

Table 342 – RPCVersionFack

Value	Meaning	Comment
0x00	Initial version	Identical coding rules
0x01	Default version	
0x02 – 0xFF	Reserved	—

4.10.3.2.24 Coding of the field RPCPad1

This field shall be coded as data type Unsigned8.

4.10.3.2.25 Coding of the field RPCWindowSize

This field shall be coded as data type Unsigned16. The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.10.3.2.26 Coding of the field RPCMaxTsdu

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.2.27 Coding of the field RPCMaxFragSize

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.2.28 Coding of the field RPCSerialNumber

This field shall be coded as data type Unsigned16. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.2.29 Coding of the field RPCSelAckLen

This field shall be coded as data type Unsigned16. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.2.30 Coding of the field RPCArrayOfSelAck

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.2.31 Coding of the field RPCDataRepresentationUUID

This field shall be coded as data type Structure containing the following elements with the value according to Table 343.

– Data1 as Unsigned32
– Data2 as Unsigned16
– Data3 as Unsigned16
– Data4 as array of eight Unsigned8

The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian) taking into account the exceptions defined in Table 327.

Table 343 – RPCDataRepresentationUUID – defined value

Value [UUID]	Value	Description
8A885D04-1CEB-11C9-9FE8-08002B104860	Data representation	Identifies the RPC data representation uniquely.

4.10.3.3 Coding section related to NDREPMapPDU

4.10.3.3.1 Coding of the field DeviceType

This field shall be coded as data type VisibleString[25]. The value shall be set by the manufacturer according to 5.2.7.1 and contain at least one non-“<Blank>” character.

It shall be used together with the OrderID as subsequent identification below the DeviceID and as a human readable identification of the device. The content of this field shall be identical to the field DeviceVendorValue using the rules stated there.

4.10.3.3.2 Coding of the field OrderID

This field shall be coded as data type VisibleString[20]. The value shall be set by the manufacturer according to 5.2.7.1.

It shall be used together with the DeviceType as subsequent address information below the DeviceID.

4.10.3.3.3 Coding of the field HWRevision

This field shall be coded as data type VisibleString[5]. The value shall be set manufacturer specific using the characters "0"-"9" and "<Blank>". The range is from "<Blank><Blank><Blank><Blank>0" to "99999".

NOTE As an example HWRevision could be "<Blank><Blank><Blank><Blank>2", "<Blank>5714", or "61214".

This field shall show the same revision than the field IM_Hardware_Revision.

4.10.3.3.4 Coding of the field SWRevisionPrefix

4.10.3.3.4.1 General

This field shall be coded as data type VisibleString[1]. The value shall be set manufacturer specific using the characters:

– "V" for an officially released version
– "R" for Revision
– "P" for Prototype
– "U" for Under Test (Field Test)
– "T" for Test Device

4.10.3.3.4.2 Submodules without software / firmware

Recommendation for submodules without software / firmware:

– Use "V" in conjunction with the field SWRevision containing "<Blank><Blank>0<Blank><Blank>0<Blank><Blank>0".
– Use "V" in conjunction with the fields IM_SWRevision_Functional_Enhancement, IM_SWRevision_Bug_Fix, and IM_SWRevision_Internal_Change containing "0x00".

4.10.3.3.5 Coding of the field SWRevision

This field shall be coded as data type VisibleString[9]. The value shall be set manufacturer specific using the character "0"-"9" and "<Blank>". The range is from "<Blank><Blank>0<Blank><Blank>0<Blank><Blank>0" to "999999999".

As an example SWRevision could be "<Blank><Blank>1<Blank><Blank>0<Blank><Blank>0". The presentation layer may insert a "." After each group of three octets for better visualization.

4.10.3.3.6 Coding of the field Blank

This field shall be coded as data type OctetString with 1 octet.

The value shall be set to 0x20.

4.10.3.3.7 Coding of the field RPCInquiryType

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The coding shall be according to Table 344.

Table 344 – RPCInquiryType

Value (hexadecimal)	Meaning	Using
0x00000000	Read all registered interfaces with objects.	Mandatory
0x00000001	Read all objects for one registered interface.	Optional
0x00000002	Read all interfaces including a dedicated object.	Optional
0x00000003	Read one dedicated interface with one dedicated object.	Optional
0x00000004 – 0xFFFFFFFF	Reserved	—

4.10.3.3.8 Coding of the field RPCObjectReference

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to 1.

4.10.3.3.9 Coding of the field RPCInterfaceReference

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to 2.

4.10.3.3.10 Coding of the field RPCInterfaceVersionMajor

This field shall be coded as data type Unsigned16.

The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian) with the exceptions defined in Table 327.

4.10.3.3.11 Coding of the field RPCInterfaceVersionMinor

This field shall be coded as data type Unsigned16.

The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian) with the exceptions defined in Table 327.

4.10.3.3.12 Coding of the field RPCVersionOption

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to 1.

4.10.3.3.13 Coding of the field RPCEntryHandleAttribute

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to zero.

4.10.3.3.14 Coding of the field RPCEntryHandleUUID

This field shall be coded as data type UUID. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.3.15 Coding of the field RPCMaxEntries

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set at least to 1.

4.10.3.3.16 Coding of the field RPCNumberOfEntries

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.3.17 Coding of the field RPCEntriesOffset

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to zero.

4.10.3.3.18 Coding of the field RPCEntriesCount

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.3.19 Coding of the field RPCTowerReference

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to 3.

4.10.3.3.20 Coding of the field RPCAnnotationOffset

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to zero.

4.10.3.3.21 Coding of the field RPCAnnotationLength

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set from 0 to 64.

4.10.3.3.22 Coding of the field EndTerm

This field shall be coded as data type Unsigned8. The value shall be set to zero.

4.10.3.3.23 Coding of the field RPCGap

This field shall be coded as data type Unsigned8.

4.10.3.3.24 Coding of the field RPCTowerLength

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.3.25 Coding of the field RPCTowerOctetStringLength

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

4.10.3.3.26 Coding of the field RPCEPMapStatus

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The coding shall be according to Table 345.

Table 345 – RPCEPMapStatus

Value (hexadecimal)	Meaning
0x00000000	RPC okay
0x16C9A0D6	Endpoint not registered
others	Reserved

4.10.3.3.27 Coding of the field RPCFloorCount

This field shall be coded as data type Unsigned16. The encoding shall be independently of the field RPCDRep always little endian.

The value shall be set to 5.

4.10.3.3.28 Coding of the field RPCLHSByteCount

This field shall be coded as data type Unsigned16. The encoding shall be independently of the field RPCDRep always little endian.

4.10.3.3.29 Coding of the field RPCRHSByteCount

This field shall be coded as data type Unsigned16. The encoding shall be independently of the field RPCDRep always little endian.

4.10.3.3.30 Coding of the field RPCID

This field shall be coded as data type Unsigned8.

The value shall be set to 0x0D.

4.10.3.3.31 Coding of the field RPCProtocolIdentifier

This field shall be coded as data type Unsigned8.

The value shall be set to 0x0A.

4.10.3.3.32 Coding of the field RPCServerUDPPort

This field shall be coded as data type Unsigned8.

The value shall be set to 0x08.

4.10.3.3.33 Coding of the field RPCHostAddress

This field shall be coded as data type Unsigned8.

The value shall be set to 0x09.

4.10.3.3.34 Coding of the field RPCPortNumber

This field shall be coded as data type Unsigned16. The encoding shall be independently of the field RPCDRep always big endian.

4.10.3.3.35 Coding of the field RPCIPAddress

This field shall be coded as data type Unsigned32. The encoding shall be independently of the field RPCDRep always big endian.

The value may be zero.

4.10.3.4 Coding section related to NDRData

4.10.3.4.1 Coding of the field ArgsMaximum

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to the maximum buffer size available for the response, and in addition cover the buffer size for the request.

4.10.3.4.2 Coding of the field ArgsLength

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to the number of octets within the PROFINETIOServiceReqPDU or PROFINETIOServiceResPDU.

4.10.3.4.3 Coding of the field MaximumCount

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

In case of a request the value shall be set to the same value as in the field ArgsMaximum. Within the response the value shall be taken from the field ArgsMaximum of the appropriate request.

In case of an inconsistency in the coding of the “Uni-dimensional Conformant-varying Array” the packet should be rejected at the server side and an error should be reported locally at the client side, as stated in The Open Group – Publication C706.

4.10.3.4.4 Coding of the field Offset

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to zero.

4.10.3.4.5 Coding of the field ActualCount

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The value shall be set to the same value as in the field ArgsLength.

4.10.3.4.6 Coding of the field PNIOStatus

This field shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian) within the first field of the NDRDataResponse. In all other cases the octet ordering shall be big endian.

The content is defined in 5.2.6. The PNIOStatus shall be calculated according to Formula (44).

$$\begin{aligned} \text{PNIOStatus} = {} & \text{ErrorCode} \times 16\,777\,216 + \\ & \text{ErrorDecode} \times 65\,536 + \\ & \text{ErrorCode1} \times 256 + \\ & \text{ErrorCode2} \end{aligned} \tag{44}$$

where

ErrorCode	is the error code value
ErrorDecode	is the error decode value
ErrorCode1	is the error code1 value
ErrorCode2	is the error code2 value

4.10.3.5 Coding section related to RPC (NCA Codes)

4.10.3.5.1 Coding of the field RPCNCAFaultStatus

The RPC specific NCA error codes shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The values shall be according to Table 346.

Table 346 – Values of NCAFaultStatus

Value (hexadecimal)	Definition
0x1C000001	NCA_s_fault_int_div_by_zero
0x1C000002	NCA_s_fault_addr_error
0x1C000003	NCA_s_fault_fp_div_zero
0x1C000004	NCA_s_fault_fp_underflow
0x1C000005	NCA_s_fault_fp_overflow
0x1C000006	NCA_s_fault_invalid_tag
0x1C000007	NCA_s_fault_invalid_bound
0x1C000008	NCA_s_rpc_version_mismatch
0x1C000009	NCA_s_unspec_reject
0x1C00000A	NCA_s_bad_actid
0x1C00000B	NCA_s_who_are_you_failed

Value (hexadecimal)	Definition
0x1C00000C	NCA_s_manager_not_entered
0x1C00000D	NCA_s_fault_chancel
0x1C00000E	NCA_s_fault_ill_inst
0x1C00000F	NCA_s_fault_fp_error
0x1C000010	NCA_s_fault_int_overflow
0x1C000012	NCA_s_fault_unspec
0x1C000013	NCA_s_fault_remote_comm_failure
0x1C000014	NCA_s_fault_pipe_empty
0x1C000015	NCA_s_fault_pipe_closed
0x1C000016	NCA_s_fault_pipe_order
0x1C000017	NCA_s_fault_pipe_discipline
0x1C000018	NCA_s_fault_pipe_comm_error
0x1C000019	NCA_s_fault_pipe_memory
0x1C00001A	NCA_s_fault_context_mismatch
0x1C00001B	NCA_s_fault_remote_no_memory
0x1C00001C	NCA_s_invalid_pres_context_id
0x1C00001D	NCA_s_unsupported_authn_level
0x1C00001F	NCA_s_invalid_checksum
0x1C000020	NCA_s_invalid_crc
0x1C000021	NCA_s_fault_user_defined
0x1C000022	NCA_s_fault_tx_open_failed
0x1C000023	NCA_s_fault_codeset_conv_error
0x1C010001	NCA_s_comm_failure
0x1C010015	NCA_s_fault_string_too_long

4.10.3.5.2 Coding of the field RPCNCARejectStatus

The RPC specific NCA error codes shall be coded as data type Unsigned32. The octet ordering shall be according to the value of the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The values shall be according to Table 347.

Table 347 – Values of NCARejectStatus

Value (hexadecimal)	Definition
0x1C000008	NCA_rpc_version_mismatch
0x1C000009	NCA_unspec_reject
0x1C00000A	NCA_s_bad_actid
0x1C00000B	NCA_who_are_you_failed
0x1C00000C	NCA_manager_not_entered
0x1C010002	NCA_op_rng_error
0x1C010003	NCA_unk_if
0x1C010006	NCA_wrong_boot_time
0x1C010009	NCA_s_you_crashed
0x1C01000B	NCA_proto_error
0x1C010013	NCA_out_args_too_big
0x1C010014	NCA_server_too_busy
0x1C010017	NCA_unsupported_type
0x1C00001D	NCA_unsupported_authn_level
0x1C00001F	NCA_invalid_checksum
0x1C000020	NCA_invalid_crc

4.10.4 FAL Service Protocol Machines

There is no FAL Service Protocol Machine (FSPM) defined for this Protocol.

4.10.5 Application Relationship Protocol Machines

4.10.5.1 RPC Protocol Machine

4.10.5.1.1 Primitive definitions

4.10.5.1.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by RPC are described in the service definition and shown in Table 348.

Table 348 – Remote primitives issued or received by RPC

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.10.5.1.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by RPC are described in the service definition and shown in Table 349.

Table 349 – Local primitives issued or received by RPC

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.10.5.1.2 State transition diagram

The state transition diagram of RPC is defined in the Technical standard DCE1.1.

4.10.5.1.3 State machine description

The state machine description is defined in the Technical standard DCE1.1.

4.10.5.1.4 RPC state table

The state table is defined in the Technical standard DCE1.1.

4.10.5.1.5 Functions, Macros, Timers and Variables

The functions are defined in the Technical standard DCE1.1.

4.10.5.2 Monitoring of services

The Ping service initiated by a service requester is called to monitor the health of the responder. The Cancel service may be repeated up to three times.

Timeout Ping
The value shall be 2 s.

NOTE The ping service is used if the request is completely sent and the response is still pending.

Timeout FRAG
The value shall be 2 s.

NOTE The transmission of a fragment will be repeated up to three times after 2 s if there is no acknowledge.

Timeout Cancel
The value shall be 1 s.

NOTE The Cancel service will be repeated up to three times after 1 s if there is no reaction.

Timeout Resend, Timeout Ack, Timeout Broadcast, Timeout IDLE, Timeout WAIT
The values are not used.

NOTE The above described parameters are defined with their values in DCE RPC.

4.10.6 DLL Mapping Protocol Machines

There is no DLL Mapping Protocol Machine (DMPM) defined for this Protocol.

4.11 Link layer discovery

4.11.1 General

This document uses link layer discovery according to IEEE Std 802.1AB-2016, 11.3.1 in version 1 mode together with version 1 MIB. The additional support of version 2 mode and version 2 MIBs is optional.

The IEEE Std 802.1AB-2016, 9.2.5.20 defined variable txNow := TRUE is used concurrently with any change of the fields of the LLDP-PDU. In this case, a node transmits LLDP PDUs “on data change”.

NOTE IEEE Std 802.1AB-2016, 9.2.7.8 defined function “somethingChangedLocal()” initiates a LLDP PDU transmission.

4.11.2 FAL common syntax description

4.11.2.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.11.2.2 LLDP APDU abstract syntax

Table 350 defines the abstract syntax of the LLDP PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 350 shall represent the content of the DLSDU in Table 10.

Table 350 – LLDP APDU syntax

APDU name	APDU structure
LLDP-PDU	LLDPChassis, LLDPPort, LLDPTTL, LLDPSystemCapabilities, LLDP-PNIO-PDU, LLDPEnd
LLDP-PNIO-PDU	{ [LLDP_PNIO_DELAY] [a], LLDP_PNIO_PORTSTATUS, [LLDP_PNIO_ALIAS], [LLDP_PNIO_MRPPORTSTATUS] [b], LLDP_PNIO_CHASSIS_MAC, LLDP8023MACPHY [d], LLDP8023MaxFrameSize [i], LLDP8023AddEthCap [l], LLDPManagement, [LLDP_PNIO_PTCPSTATUS] [c], [LLDP_PNIO_MAUTypeExtension] [f], [LLDP_PNIO_MRPICPORTSTATUS] [g], [LLDP_PNIO_NMEDomainUUID] [h], [LLDP_PNIO_NMENameUUID] [j], [LLDP_PNIO_NMEParameterUUID] [j], [LLDP_PNIO_NMEManagementAddr] [j], [LLDP_PNIO_1ASWorkingClock] [k], [LLDP_PNIO_1ASGlobalTime] [k], [LLDPOption*] [e], [LLDP8021*], [LLDP8023*] }

[a] Shall only exist, if LineDelay measurement is supported.

[b] Shall only exist, if MRP is activated for this port.

[c] Shall only exist, if PTCP is activated by means of the PDSyncData Record.

[d] Shall only exist, if IEEE Std 802.3 is used.

[e] Other LLDP options may be used concurrently.

[f] Shall exist, if a MAUType with MAUTypeExtension is used, and may exist otherwise.

[g] Shall only exist, if MRP Interconnection is activated for this port.

[h] Shall only exist, if a NME domain name is assigned to this device.

[i] Shall only exist, if this MAC/PHY supports envelope frames.

[j] Shall only exist, if CIMNetConfDataAdjust is assigned to this device by an NME.

[k] Shall only exist, if the target clock of the sender is in sync.

[l] Shall exist, if preemption is supported, and may exist otherwise.

Table 351 defines structures for substitutions of elements of the APDU structures shown in Table 350.

Table 351 – LLDP substitutions

Substitution name	Structure
LLDPChassis with MultipleInterfaceMode.-NameOfDevice:=0	LLDPChassisStationName ^ LLDPChassisMacAddress [a]
LLDPChassis with MultipleInterfaceMode.-NameOfDevice:=1	LLDPChassisStationName ^ LLDPChassisMacAddress [c]
LLDPChassisStationName	LLDP_TLVHeader [b], LLDP_ChassisIDSubType(7) [b], LLDP_ChassisID
LLDPChassisMacAddress	LLDP_TLVHeader [b], LLDP_ChassisIDSubType(4) [b], (CMResponderMacAdd ^ CMInitiatorMacAdd) [d]
LLDPPort	LLDP_TLVHeader [b], LLDP_PortIDSubType(7) [b], LLDP_PortID
LLDPSystemCapabilities	LLDP_TLVHeader [b], LLDP_SystemCapabilities [b k], LLDP_EnabledCapabilities [b k]
LLDPTTL	LLDP_TLVHeader [b], LLDP_TimeToLive(20) [b]
LLDP_PNIO_Header	LLDP_TLVHeader [b], LLDP_OUI(00-0E-CF)
LLDP_PNIO_DELAY	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x01), PTCP_PortRxDelayLocal, PTCP_PortRxDelayRemote, PTCP_PortTxDelayLocal, PTCP_PortTxDelayRemote, CableDelayLocal
LLDP_PNIO_PORTSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x02), RTClass2_PortStatus, RTClass3_PortStatus
LLDP_PNIO_ALIAS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x03), AliasNameValue
LLDP_PNIO_MRPPORTSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x04), MRP_DomainUUID, MRRT_PortStatus
LLDP_PNIO_CHASSIS_MAC	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x05), (CMResponderMacAdd ^ CMInitiatorMacAdd) [d]
LLDP_PNIO_PTCPSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x06), PTCP_MasterSourceAddress [e], PTCP_SubdomainUUID [f], IRDataUUID [g], LLDP_LengthOfPeriod [g], LLDP_RedOrangePeriodBegin [g], LLDP_OrangePeriodBegin [g], LLDP_GreenPeriodBegin [g]
LLDP_PNIO_MAUTypeExtension	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x07), MAUTypeExtension
LLDP_PNIO_MRPICPORT-STATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x08), MRPIC_DomainID, MRPIC_Role, MRPIC_MICPosition
LLDP_PNIO_NMEDomainUUID	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x09), NMEDomainUUID
LLDP_PNIO_-NMEManagementAddr	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x0A), NMEManagementAddr
LLDP_PNIO_NMENameUUID	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x0B), NMENameUUID
LLDP_PNIO_NMEParameterUUID	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x0C), NMEParameterUUID
LLDP_PNIO_1ASWorkingClock	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x0D), TimeDomainNumber, TimeDomainMasterIdentity
LLDP_PNIO_1ASGlobalTime	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x0E), TimeDomainNumber, TimeDomainMasterIdentity
LLDPEnd	LLDP_TLVHeader [b](0)
LLDP8021	LLDP_TLVHeader [b], LLDP_OUI(00-80-C2) [h], LLDP_8021_SubType [h], Data [h]
LLDP8023	LLDP_TLVHeader [b], LLDP_OUI(00-12-0F) [i], LLDP_8023_SubType [i], Data [i]
LLDP8023MACPHY	LLDP_TLVHeader [b], LLDP_OUI(00-12-0F) [i], LLDP_8023_SubType(1) [i], LLDP_8023_AUTONEG [i], LLDP_8023_PMDCAP [i], LLDP_8023_OPMAU [i]
LLDP8023MaxFrameSize	LLDP_TLVHeader [b], LLDP_OUI(00-12-0F) [i], LLDP_8023_SubType(4) [i], LLDP_8023_MaximumFrameSize [i]
LLDP8023AddEthCap	LLDP_TLVHeader [b], LLDP_OUI(00-12-0F) [i], LLDP_8023_SubType(7) [i], LLDP_8023_AdditionalEthernetCapabilities [i]

Substitution name	Structure
LLDPManagement	LLDP_TLVHeader [b], LLDP_ManagementData [j]

[a] LLDPChassisMacAddress shall be used if no NameOfStation is assigned.

[b] The encoding of the fields shall be according to IEEE Std 802.1AB.

[c] LLDPChassisMacAddress shall only be used if the NameOfDevice is equal to the NameOfStation and no NameOfStation is assigned.

[d] Shall be the interface MAC address of the transmitting node.

[e] Shall be set to zero, if unknown.

[f] Shall be PTCP_SubdomainUUID of PTCP_SyncID := 0. Otherwise the value shall be zero.

[g] Shall be set to zero, if unknown.

[h] Shall be set according to IEEE Std 802.1Q-2018, Annex D.

[i] Shall be set according to IEEE Std 802.3-2018, Clause 79.

[j] Shall be set according to IEEE Std 802.1AB, 8.5.9. It is recommended to set the object identifier according to 4.16.5 or with a vendor specific MIB. If both are not implemented, then the “object identifier (OID) string length” shall be set to zero.

[k] The values "Station Only" or "C-VLAN component" according to IEEE Std 802.1AB-2016, 8.5.8 are expected to be used for this document.

NOTE There are different kinds of MAC addresses: the port MAC address used as SourceAddress and the interface MAC address used as CMResponderMacAdd or CMInitiatorMacAdd.

4.11.3 LLDP transfer syntax

4.11.3.1 Coding of the field LLDP_PNIO_SubType

This field shall be coded as data type Unsigned8 and shall be set according to Table 352.

Table 352 – LLDP_PNIO_SubType

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Measured delay values
0x02	Port Status
0x03	Alias
0x04	MRP Ring Port Status
0x05	Interface MAC address
0x06	PTCP Status
0x07	MAUType extension
0x08	MRP Interconnection Port Status
0x09	NME domain identification
0x0A	NME Management address
0x0B	NME Name UUID
0x0C	NME Parameterization UUID
0x0D	IEEEE 802.1AS Working clock domain
0x0E	IEEEE 802.1AS Global time domain
0x0F – 0xFF	Reserved

4.11.3.2 Coding of the field PTCP_PortRxDelayLocal

This field shall be coded as data type Unsigned32 with values according to Table 353. The time base shall be one nanosecond.

Table 353 – PTCP_PortRxDelayLocal

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Local RX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.11.3.3 Coding of the field PTCP_PortRxDelayRemote

This field shall be coded as data type Unsigned32 with values according to Table 354. The time base shall be one nanosecond.

Table 354 – PTCP_PortRxDelayRemote

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Remote RX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.11.3.4 Coding of the field PTCP_PortTxDelayLocal

This field shall be coded as data type Unsigned32 with values according to Table 355. The time base shall be one nanosecond.

Table 355 – PTCP_PortTxDelayLocal

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Local TX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.11.3.5 Coding of the field PTCP_PortTxDelayRemote

This field shall be coded as data type Unsigned32 with values according to Table 356. The time base shall be one nanosecond.

Table 356 – PTCP_PortTxDelayRemote

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Remote TX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.11.3.6 Coding of the field CableDelayLocal

This field shall be coded as data type Unsigned32 with values according to Table 357. The time base shall be one nanosecond.

Table 357 – CableDelayLocal

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x000FFFFF	Measured cable delay
0x00100000 – 0xFFFFFFFF	Reserved

4.11.3.7 Coding of the field RTClass2_PortStatus

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: RTClass2_PortStatus.State
This field shall be set according to Table 358.

Table 358 – RTClass2_PortStatus.State

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used
other	Reserved	—

Bit 2 – 15: RTClass2_PortStatus.Reserved
This field shall be set according to 3.4.2.2.

4.11.3.8 Coding of the field RTClass3_PortStatus

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 2: RTClass3_PortStatus.State
This field shall be set according to Table 359.

Table 359 – RTClass3_PortStatus.State

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used or configured if LLDP_RedOrangePeriodBegin.Valid equals 1. RED period is deactivated.
0x01	Reserved	—
0x02	RTCLASS3_UP	RED period activated for transmission of RT_CLASS_3 Frames. Expected next state: RTCLASS3_RUN
0x03	Reserved	—
0x04	RTCLASS3_RUN	RED period activated for transmission and reception of RT_CLASS_3 Frames
0x05 – 0x07	Reserved	—

Bit 3 – 11: RTClass3_PortStatus.Reserved
This field shall be set according to 3.4.2.2.

Bit 12: RTClass3_PortStatus.Fragmentation
This field shall be set according to Table 360.

Table 360 – RTClass3_PortStatus.Fragmentation

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default The fragmentation mode for this port is disabled.
0x01	ON	The fragmentation mode for this port is enabled.

The information whether the fragmentation is activated shall be derived from FrameDataProperties.FragmentationMode and PDIRGlobalData.YellowTime.

Bit 13: RTClass3_PortStatus.PreambleLength
This field shall be set according to Table 361.

Table 361 – RTClass3_PortStatus.PreambleLength

Value (hexadecimal)	Meaning	Usage
0x00	Seven octets	Default The PHY uses seven octets preamble when transmitting
0x01	One octet	The PHY uses one octet preamble when transmitting

It is recommended, that a PHY / MAC supports one octet preamble length when sending using the MAUType 100BaseTXFD.

A PHY / MAC should support one octet preamble length when receiving using the MAUType 100BaseTXFD.

Table 362 contains the truth table used for the shortening of the preamble.

Table 362 – Truth table for shortening of the preamble

Input		Output
RTC3PSM	AdjustPreambleLength. PreambleLength.Length	RTClass3_PortStatus. PreambleLength
EXPECTED_REMOTE(LLDP_ChassisID) == REMOTE(LLDP_ChassisID) AND EXPECTED_REMOTE(LLDP_PortID) == REMOTE(LLDP_PortID) AND EXPECTED_REMOTE(IRDataUUID) == REMOTE(LLDP_IRDataUUID) AND EXPECTED_LOCAL(MAUType) == LOCAL(MAUType)	Seven Octets	Seven Octets
	One Octet	One Octet

Input		Output
RTC3PSM	**AdjustPreambleLength. PreambleLength.Length**	**RTClass3_PortStatus. PreambleLength**
! (EXPECTED_REMOTE(LLDP_ChassisID) == REMOTE(LLDP_ChassisID) AND EXPECTED_REMOTE(LLDP_PortID) == REMOTE(LLDP_PortID) AND EXPECTED_REMOTE(IRDataUUID) == REMOTE(LLDP_IRDataUUID) AND EXPECTED_LOCAL(MAUType) == LOCAL(MAUType))	—	Seven Octets

Bit 14: RTClass3_PortStatus.Reserved
This field shall be set according to 3.4.2.2.

Bit 15: RTClass3_PortStatus.Optimized
This field shall be set according to Table 363.

Table 363 – RTClass3_PortStatus.Optimized

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	ON	Reserved

4.11.3.9 Coding of the field MRRT_PortStatus

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: MRRT_PortStatus.State
This field shall be set according to Table 364.

Table 364 – MRRT_PortStatus.State

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used / not configured
0x01 – 0x03	Reserved	—

Bit 2 – 15: MRRT_PortStatus.Reserved
This field shall be set according to 3.4.2.2.

4.11.3.10 Coding of the field IRDataUUID

This field shall be coded as data type UUID according to Table 365.

Table 365 – IRDataUUID

Value (UUID)	Meaning	Usage
00000000-0000-0000-0000-000000000000	No RT_CLASS_3 domain	Initial value
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFE	UUID for a RT_CLASS_3 domain	Generated by engineering and used by IO controller and IO device to identify a RT_CLASS_3 domain

4.11.3.11 Coding of the field LLDP_RedOrangePeriodBegin

As defined for RT_CLASS_3 the usage of a port is divided into different time periods. Also the usage of a port is divided into transmit and receive direction. The coding of this field shall be according to 3.4.2.3.5 and to Figure 68. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_RedOrangePeriodBegin.Offset
This field shall be set according to Table 366.

Table 366 – LLDP_RedOrangePeriodBegin.Offset

Value (hexadecimal)	Meaning	Usage
0x00000000 – 0x003D08FF	Offset relative to the begin of the cycle in nanoseconds	Begin of the RT_CLASS_3 period of the transmit direction of the port. If a RedPeriod is defined, the value shall be derived from the minimum of RedOrangePeriodBegin, else the value shall be derived from ReservedIntervalBegin.
0x003D0900 – 0x7FFFFFFF	Reserved	—

Bit 31: LLDP_RedOrangePeriodBegin.Valid
This field shall be set according to Table 367.

Table 367 – LLDP_RedOrangePeriodBegin.Valid

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	The value of LLDP_RedOrangePeriodBegin.Offset is not valid. It shall be set to zero.
0x01	Valid	The value of LLDP_RedOrangePeriodBegin.Offset is valid.

4.11.3.12 Coding of the field LLDP_OrangePeriodBegin

The coding of this field shall be according to 3.4.2.3.5 and to Figure 68. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_OrangePeriodBegin.Offset
This field shall be set according to Table 368.

Table 368 – LLDP_OrangePeriodBegin.Offset

Value (hexadecimal)	Meaning	Usage
0x00000000 – 0x003D08FF	Reserved	—
0x003D0900 – 0x7FFFFFFF	Reserved	—

Bit 31: LLDP_OrangePeriodBegin.Valid
This field shall be set according to Table 369.

Table 369 – LLDP_OrangePeriodBegin.Valid

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	LLDP_OrangePeriodBegin.Offset is not valid. It shall be set to zero.
Other	—	Reserved

4.11.3.13 Coding of the field LLDP_GreenPeriodBegin

As defined for RT_CLASS_1, RT_CLASS_2, RT_CLASS_UDP and the other protocols the usage of a port is divided into different time periods. Also, the usage of a port is divided into transmit and receive. The coding of this field shall be according to 3.4.2.3.5 and to Figure 68. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_GreenPeriodBegin.Offset
This field shall be set according to Table 370.

Table 370 – LLDP_GreenPeriodBegin.Offset

Value (hexadecimal)	Meaning	Usage
0x00000000 – 0x003D08FF	Offset relative to the begin of the cycle in nanoseconds	Begin of the unrestricted period of the transmit direction of the port. The value shall be derived from ReservedIntervalEnd.
0x003D0900 – 0x7FFFFFFF	Reserved	—

Bit 31: LLDP_GreenPeriodBegin.Valid
This optional field shall be set according to Table 371.

Table 371 – LLDP_GreenPeriodBegin.Valid

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	The value of LLDP_GreenPeriodBegin.Offset is not valid. It shall be set to zero.
0x01	Valid	The value of LLDP_GreenPeriodBegin.Offset is valid.

4.11.3.14 Coding of the field LLDP_LengthOfPeriod

A port is divided into different time periods. The duration of all periods is shown by this field. The coding of this field shall be according to 3.4.2.3.5 and Figure 68. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_LengthOfPeriod.Length
This field shall be set according to Table 372.

Table 372 – LLDP_LengthOfPeriod.Length

Value (hexadecimal)	Meaning	Usage
0	—	May be used together with LLDP_LengthOfPeriod.Valid := 0
Other	Reserved	—
0x00007A12 – 0x003D0900	Duration of a cycle in nanoseconds	The value shall be a multiple of 31 250 ns. See 5.2.4.58
0x003D0900 – 0x7FFFFFFF	Reserved	—

Bit 31: LLDP_LengthOfPeriod.Valid
This optional field shall be set according to Table 373.

Table 373 – LLDP_LengthOfPeriod.Valid

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	LLDP_LengthOfPeriod.Length is not valid. It shall be set to zero.
0x01	Valid	LLDP_LengthOfPeriod.Length is valid.

4.11.3.15 Coding of the field LLDP_POINT

This field shall be coded as data type OctetString with one octet. The value shall be ".".

NOTE The field LLDP_POINT is not terminated by zero.

4.11.3.16 Coding of the field NMEManagementAddr

This field shall be coded according to IEEE Std 802.1AB-2016, Figure 8-11 fields "management address string length", "management address subtype", and "management address" and contain the address of the NME acting as domain manager.

It contains the information provided by the field NMENameAddress.

4.11.3.17 Coding of the field TimeDomainMasterIdentity

This field shall be coded as OctetString with eight octets according to the data type ClockIdentity as defined in IEEE Std 802.1AS.

4.11.3.18 Coding section related to LLDP

4.11.3.18.1 Coding of the field LLDP_ChassisID

This field shall be coded as data type OctetString according to Table 374 and Table 375. The content of this field shall uniquely identify one device, as defined by IEEE Std 802.1AB.

In case of TimeAware systems, the coding according Table 375 shall be used.

NOTE The field LLDP_ChassisID is not terminated by zero.

Table 374 – LLDP_ChassisID in conjunction with MultipleInterfaceMode.NameOfDevice == 0 and NameOfStation

NameOfStation exists	Data type	Meaning
Yes	OctetString[]	This field shall be coded with 1 to 240 octets according to 4.3.1.4.16.
No	OctetString[]	This field shall be coded according LLDPChassisMacAddress in Table 351 or Table 375.

Table 375 – LLDP_ChassisID in conjunction with MultipleInterfaceMode.NameOfDevice == 1

Data type	Meaning
OctetString[]	This field shall be coded with "SystemIdentification"

Not recommended:

A node, not intending to ever implement more than one interface, may fill the LLDP_ChassisID as in Table 374 even if MultipleInterfaceMode.NameOfDevice == 1 is set.

4.11.3.18.2 Coding of the field LLDP_PortID

This field shall be coded as data type OctetString according to Table 376, as defined by IEEE Std 802.1AB.

NOTE The field LLDP_PortID is not terminated by zero.

Table 376 – LLDP_PortID in conjunction with MultipleInterfaceMode.NameOfDevice

Multiple-InterfaceMode. NameOfDevice	Data type	Meaning
0	OctetString[8] or OctetString[14]	The field LLDP_PortID shall be coded according to 4.3.1.4.17
1	OctetString[]	The field LLDP_PortID shall be coded according to Formula (45)

$$LLDP\_PortID = NameOfPort + "." + NameOfStation \quad (45)$$

where

LLDP_PortID is the field used in LLDP to identify a port

NameOfPort is the local name of port

NameOfStation is the local name of station

4.11.3.18.3 Coding of the field LLDP_TimeToLive

This field shall be coded according to IEEE Std 802.1AB.

4.11.3.18.4 Coding of the field LLDP_TLVHeader

This field shall be coded according to IEEE Std 802.1AB.

4.11.3.18.5 Coding of the field LLDP_ChassisIDSubType

This field shall be coded according to IEEE Std 802.1AB.

4.11.3.18.6 Coding of the field LLDP_PortIDSubType

This field shall be coded according to IEEE Std 802.1AB.

4.11.3.18.7 Coding of the field LLDP_SystemCapabilities

This field shall be coded according to IEEE Std 802.1AB.

4.11.3.18.8 Coding of the field LLDP_EnabledCapabilities

This field shall be coded according to IEEE Std 802.1AB.

4.11.3.18.9 Coding of the field LLDPOption

This field shall be coded according to IEEE Std 802.1AB, 9.4.

4.11.3.18.10 Coding of the field LLDP_OUI

This field shall be coded according to IEEE Std 802.1AB.

4.11.3.18.11 Coding of the field LLDP_ManagementData

This field shall be coded according to IEEE Std 802.1AB.

4.11.3.19 Coding section related to LLDP_8021

4.11.3.19.1 Coding of the field LLDP_8021_SubType

This field shall be coded according to IEEE Std 802.1Q.

4.11.3.20 Coding section related to LLDP_8023

4.11.3.20.1 Coding of the field LLDP_8023_SubType

This field shall be coded according to IEEE Std 802.3.

4.11.3.20.2 Coding of the field LLDP_8023_AUTONEG

This field shall be coded according to IEEE Std 802.3.

4.11.3.20.3 Coding of the field LLDP_8023_PMDCAP

This field shall be coded according to IEEE Std 802.3.

4.11.3.20.4 Coding of the field LLDP_8023_OPMAU

This field shall be coded according to IEEE Std 802.3.

4.11.3.20.5 Coding of the field LLDP_8023_MaximumFrameSize

This field shall be coded according to IEEE Std 802.3.

4.11.3.20.6 Coding of the field LLDP_8023_AdditionalEthernetCapabilities

This field shall be coded according to IEEE Std 802.3.

4.11.4 FAL Service Protocol Machines

There is no FAL Service Protocol Machine (FSPM) defined for this Protocol.

4.11.5 Application Relation Protocol Machines

There is no Application Relation Protocol Machine (ARPM) defined for this Protocol.

4.11.6 DLL Mapping Protocol Machines

There is no DLL Mapping Protocol Machine (DMPM) defined for this Protocol.

4.12 End stations and bridges

4.12.1 General

The concepts according to following standards shall be applied:

– IEEE Std 802.3
– IEEE Std 802.1AB
– IEEE Std 802.1AS
– IEEE Std 802.1Q
– IEEE Std 802.1CB
– This document

The use of the “Cut through” principle for the forwarding of frames requires a late error handling to avoid frame fragments in a switched network. Thus, whenever an invalid frame (by FCS or SFCRC16) is detected which is already partially forwarded, it should be shortened according to the rules of the IEEE Std 802.3 at the transmitting port.

Figure 100 shows the integration of IEEE Std 802 into the model used in this document.

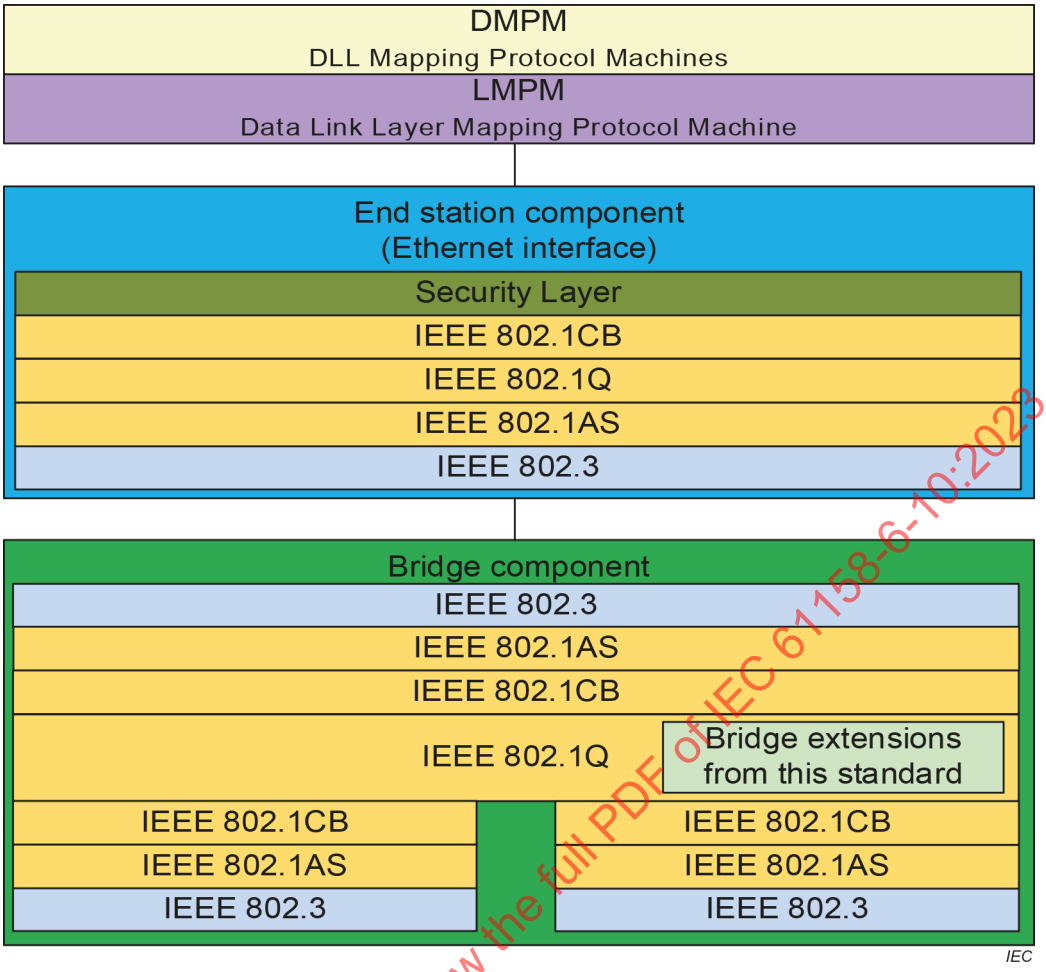


Figure 100 – DLL Mapping Protocol Machines (DMPM)

The IEEE Std 802.1Q added time awareness to its working principle and thus extended its bridge model to the end stations.

This document specifies the following three kinds of devices:

- End station
  - Devices implementing end station components
- Bridge or bridged end station
  - Devices implementing bridge components and end station components (hosting the management entity) in one housing
- Bridged end station with two ports
  - Devices implementing bridge components and end station components; with two external visible bridge component ports

Figure 101 shows the whole transmission path from Controller application to Device application(s) and back. The blue and red arrows show the contributions to the end-to-end latency, respectively.

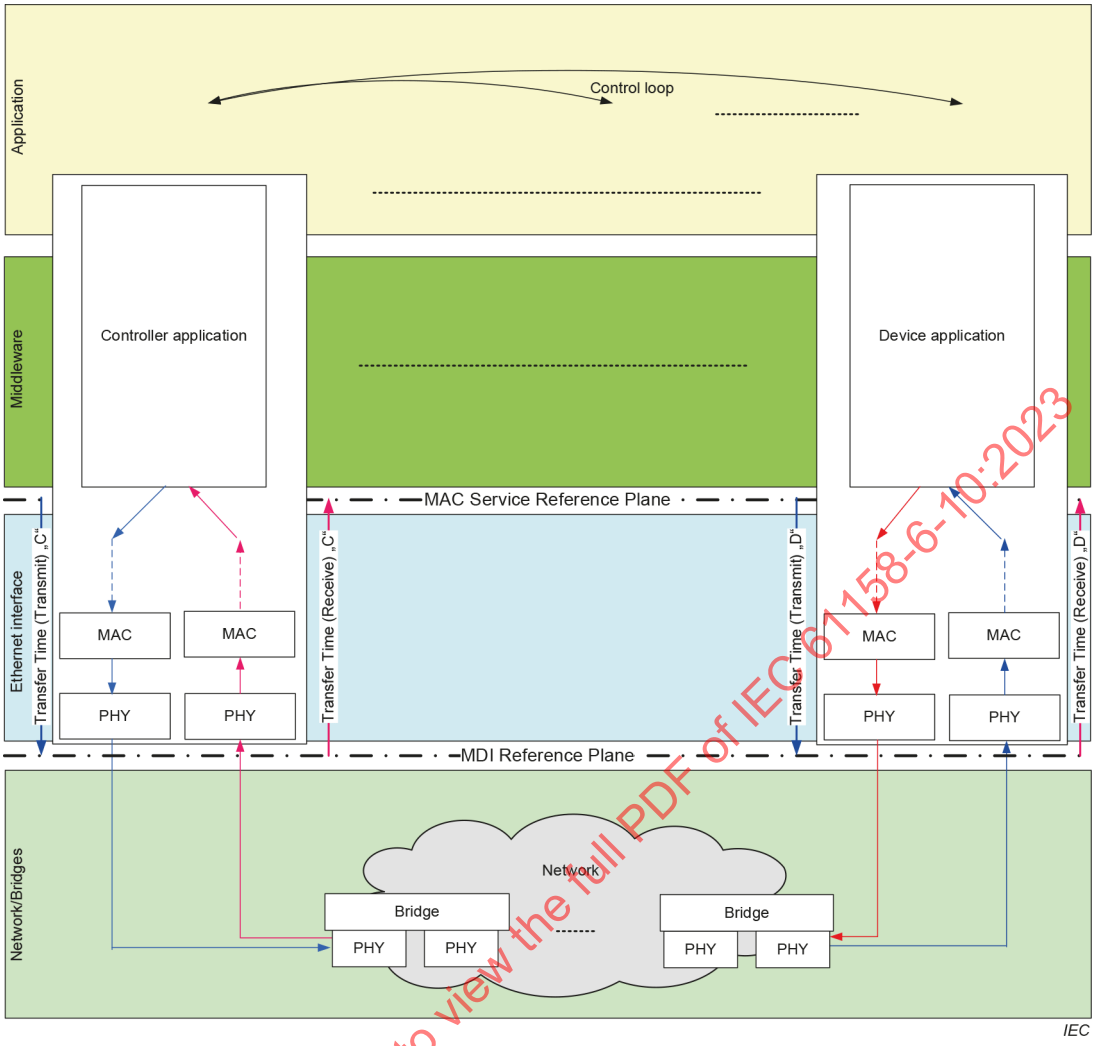


Figure 101 – Schematic diagram of data flow of control loop

Figure 101 shows the four involved levels:

– Application,
– Middleware,
– End station / Ethernet interface model, and
– Bridging / Network Forwarding.

4.12.2 Traffic classes

4.12.2.1 General

Table 377 shows the traffic class model based on IEEE Std 802.1Q according to this document.

Table 377 – Traffic classes

Name	Abbreviation	Meaning
Stream High	HIGH / ISO	Traffic category: time-aware stream Periodic executed traffic, defined deadline, zero [a] congestion loss, seamless redundancy, assigned path Synchronized network access in end stations Example: Real time data exchange for applications relying on deadlines, e.g. motion applications [a] Guaranteed due to NME/CNC means
Stream Low	LOW / CYC	Traffic category: time-aware stream Periodic executed traffic, zero [a] congestion loss, seamless redundancy, assigned path Synchronized network access in end stations Example: Real time data exchange for applications with relaxed deadline requirements [a] Guaranteed due to NME/CNC means
Stream RT	RT / A	Traffic category: stream Periodic executed traffic, assigned bandwidth [a], learned path (e.g. spanning tree) Unsynchronized network access in end stations (synchronized for optimization and to reduce possibility of congestion loss) Example: Real time data exchange for applications with relaxed deadline requirements [a] Goal is the avoidance of congestion loss
Network Control	NW	Traffic category: traffic engineered non-stream Sporadic executed traffic, assigned bandwidth [a], learned path (e.g. spanning tree) Unsynchronized network access with bandwidth limitation in end stations Example: Synchronization (gPTP), Neighborhood discovery (LLDP), Spanning tree (RSTP/MSTP), … [a] Goal is the avoidance of congestion loss
Alarms and Events	EV	Traffic category: traffic engineered non-stream Sporadic executed traffic, assigned bandwidth [a], learned path (e.g. spanning tree) Unsynchronized network access with bandwidth limitation in end stations Example: Process alarms, diagnosis alarms, reporting system, … [a] Goal is the avoidance of congestion loss
Connection Management	CO	Traffic category: traffic engineered non-stream Sporadic executed traffic, assigned bandwidth [a], learned path (e.g. spanning tree) Unsynchronized network access with bandwidth limitation in end stations Example: NME domain internal initiated traffic, e.g. connection management, parameter access, diagnosis data, … [a] Goal is the avoidance of congestion loss

Name	Abbreviation	Meaning
Best effort High	BEH	Traffic category: non-stream Sporadic executed traffic, assigned bandwidth [a], learned path (e.g. spanning tree) Unsynchronized network access with bandwidth limitation in end stations Example: NME domain passthrough, NME domain external initiated traffic, e.g. NetConf, SNMP, … NME domain internal initiated traffic, e.g. HTTP, … [a] Goal is the protection of traffic engineered non-stream classes
Best effort Low	BEL	Traffic category: non-stream Sporadic executed traffic, assigned bandwidth [a], learned path (e.g. spanning tree) Unsynchronized network access with bandwidth limitation in end stations Example: NME domain passthrough, … NME domain external initiated traffic, e.g. NetConf, SNMP, … NME domain internal initiated traffic, e.g. HTTP, … [a] Goal is the protection of traffic engineered non-stream classes

4.12.2.2 Time-aware system

End station components shall support eight queues according to:

– Table 378 for time-aware end stations,
– Table 393 for time-aware end stations without time-aware streams, and
– Table 395 for time-aware end stations with time-aware streams.

Bridge components shall support eight queues according to:

– Table 439 for time-aware bridges with time-aware streams without queue masking, and
– Table 440 for time-aware bridges with time-aware streams and queue masking.

Table 378 – Traffic class usage for time-aware system

Name	Abbreviation	Meaning
Stream High	HIGH / ISO	RT_CLASS_STREAM, class HIGH with and without end station FRER
Stream Low	LOW / CYC	RT_CLASS_STREAM, class LOW with and without end station FRER
Stream RT	RT / A	RT_CLASS_STREAM, class RT
Network Control	NW	IEEE Std 802.1AS, IEEE Std 802.1AB, RSTP, …
Alarms and Events	EV	RTA_CLASS_1 and RTA_CLASS_UDP
Connection Management	CO	Connection management, Record services, Device access, …
Best effort High	BEH	Domain internal initiated: DCP, ARP, DHCP, DNS, SNMP, … Domain external initiated: Any protocol
Best effort Low	BEL	Domain internal initiated: E.g., HTTP, … Domain external initiated: Any protocol

4.12.2.3 Non-time-aware system

End station components shall support eight queues according to:

– Table 379 for non-time-aware end stations,
– Table 397 for non-time-aware end stations without RT_CLASS_3, and
– Table 399 for non-time-aware end stations with RT_CLASS_3.

Bridge components shall support eight queues according to:

– Table 441 for non-time-aware bridges without RT_CLASS_3, and
– Table 442 for non-time-aware bridges with RT_CLASS_3.

Table 379 – Traffic class usage for non-time-aware system

Name	Abbreviation	Meaning
Stream High	HIGH / ISO	RT_CLASS_3 (if not supported n.a.)
Stream Low	LOW / CYC	n.a.
Stream RT	RT / A	RT_CLASS_1 and RT_CLASS_UDP
Network Control	NW	PTCP, IEEE Std 802.1AS, IEEE Std 802.1AB, RSTP, …
Alarms and Events	EV	RTA_CLASS_1 and RTA_CLASS_UDP
Connection Management	CO	Connection management, Record services, Device access, …
Best effort High	BEH	Domain internal initiated: DCP, ARP, DHCP, DNS, SNMP, … Domain external initiated: Any protocol
Best effort Low	BEL	Domain internal initiated: E.g., HTTP, … Domain external initiated: Any protocol

4.12.2.4 Engineering tools

Engineering or monitoring tools shall implement traffic class usage as shown in Table 380.

Table 380 – Traffic class usage for engineering tools

Name	Abbreviation	Meaning
Best effort Low	BEL	All used protocols, e.g. Record services, Device access

4.12.3 End station

4.12.3.1 General

Subclause 4.12.3 selects features out of:

– IEEE Std 802.1Q,
– IEEE Std 802.1CB, and
– IEEE Std 802.3.

The end station is the source and the destination of communication.

Figure 102, Figure 103, Figure 65, and Figure 104 show the application model of an end station based on IEEE Std 802.1Q according to this document.

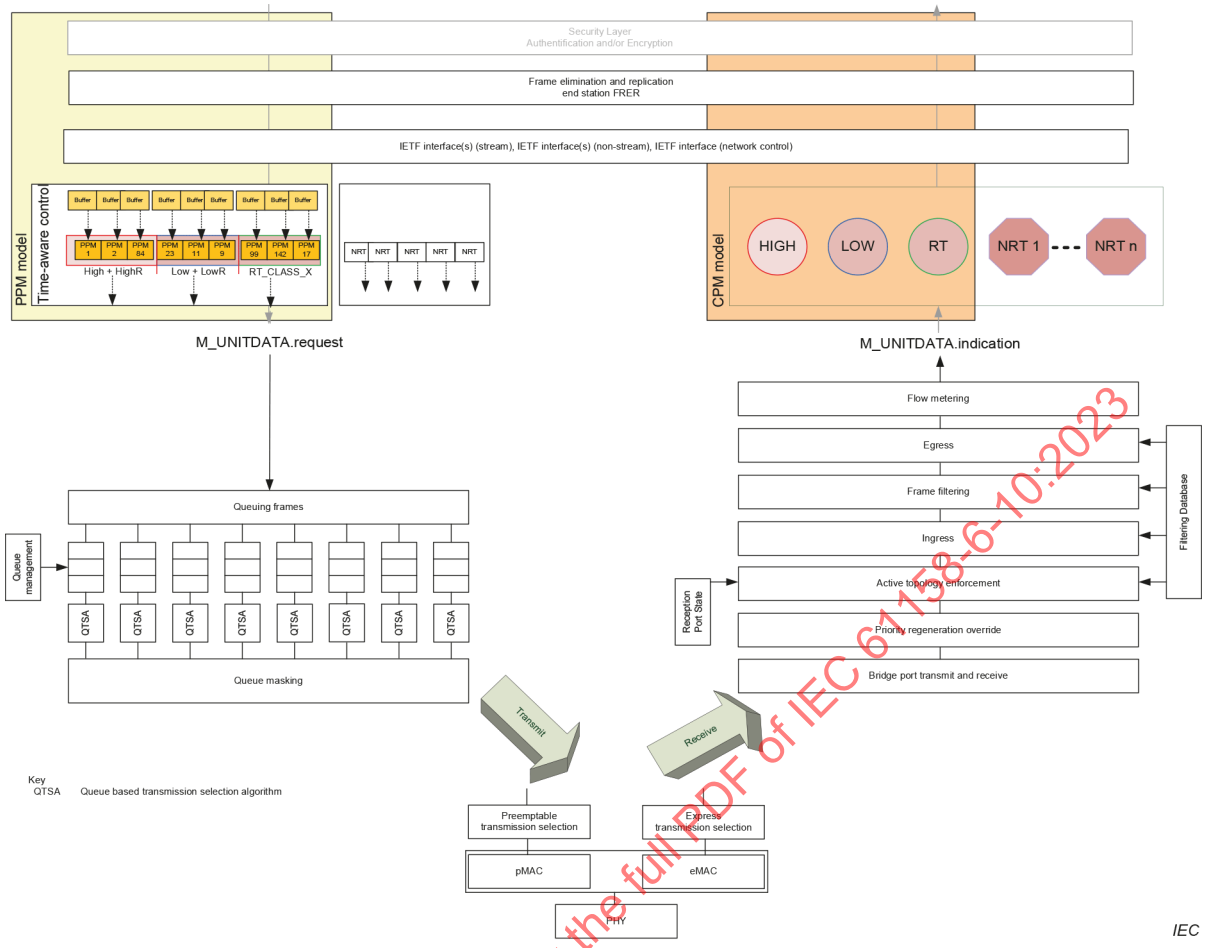


Figure 102 – End station model with IEEE Std 802.1Q alignment

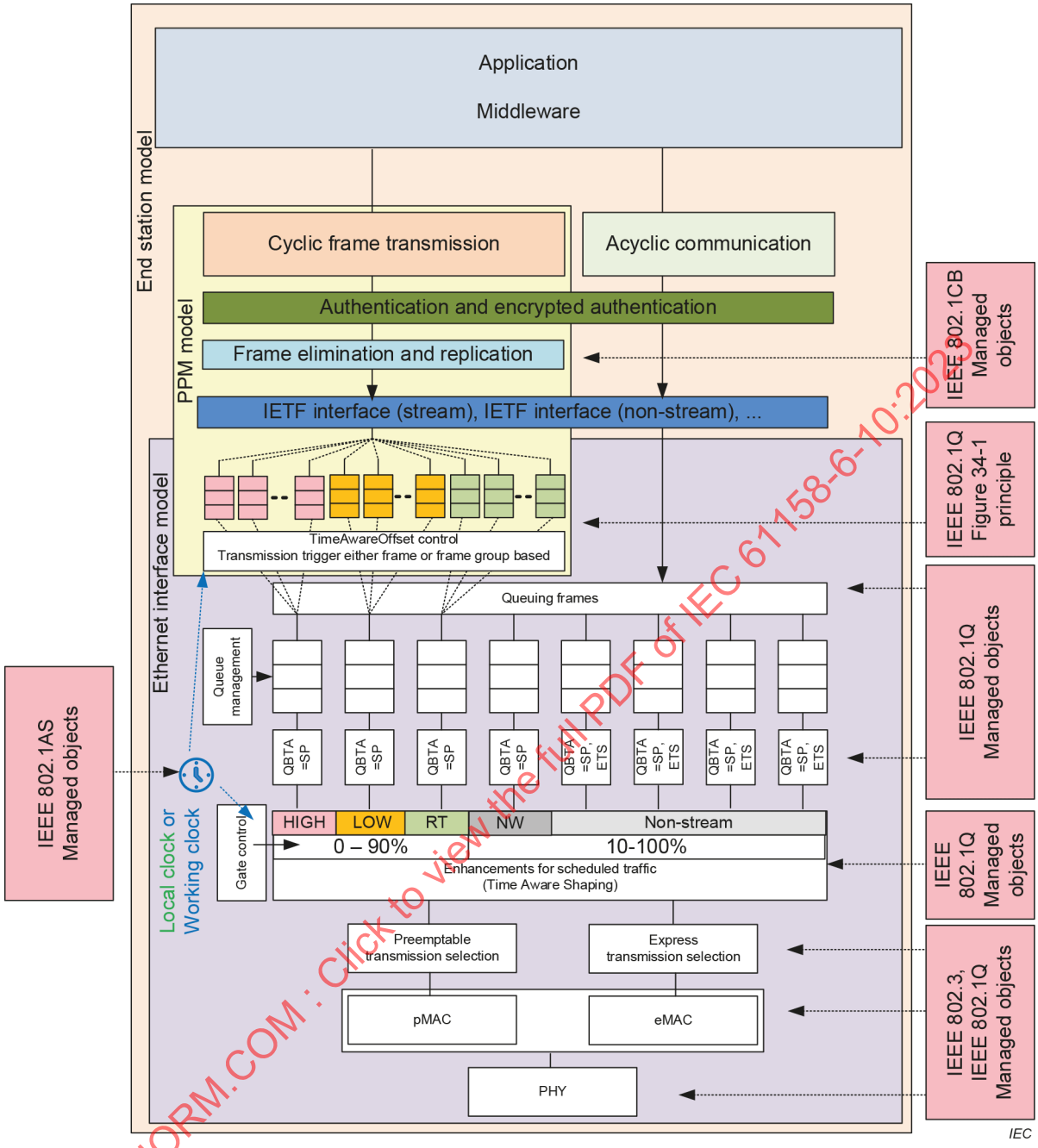


Figure 103 – Ethernet interface model with IEEE alignment – transmit direction

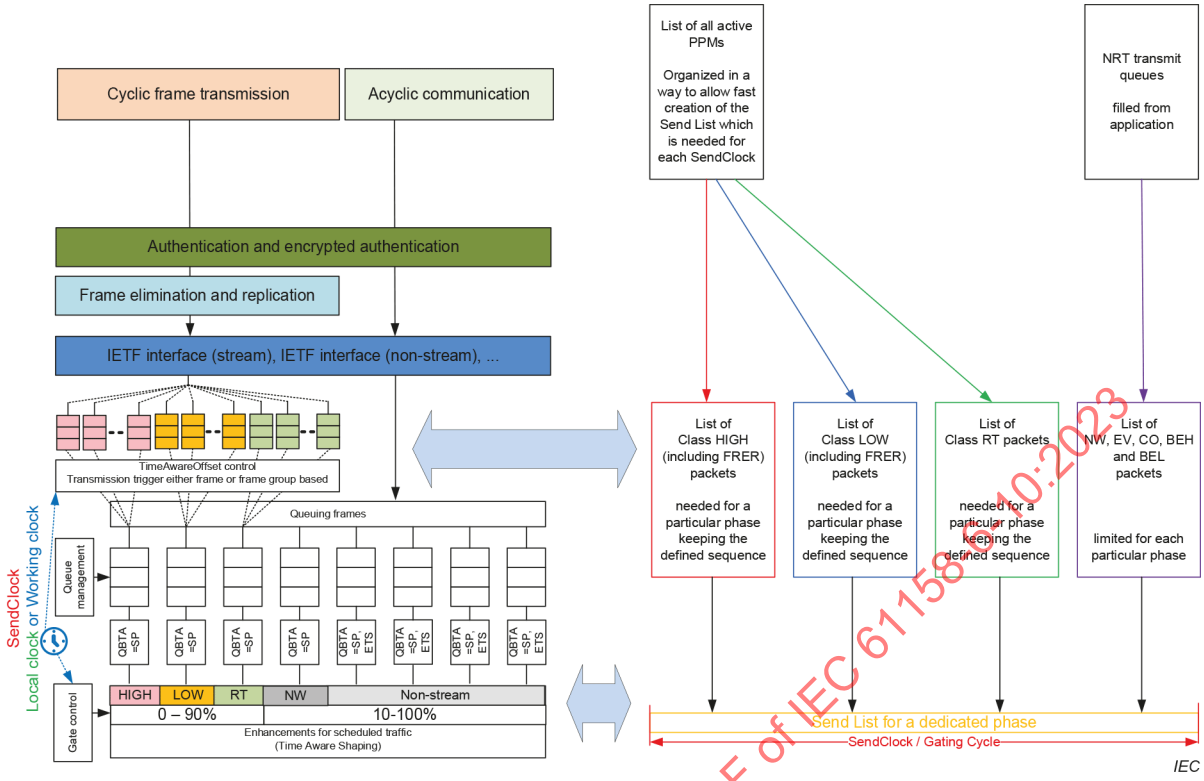


Figure 104 – SendListControl alignment with Ethernet interface model

4.12.3.2 Enhanced transmission selection to controlled non-stream information rate

Figure 105 shows the Metro Ethernet Forum (MEF), Technical Specification 10.3 specified algorithm which shall be used to implement the IEEE Std 802.1Q-2018, Clause 37 specified ETS.

This algorithm allows bandwidth sharing between the used traffic classes and thus, offers bandwidth utilization without starving the lower priorities. The use is intended for TC3 to TC0 as shown in Table 387.

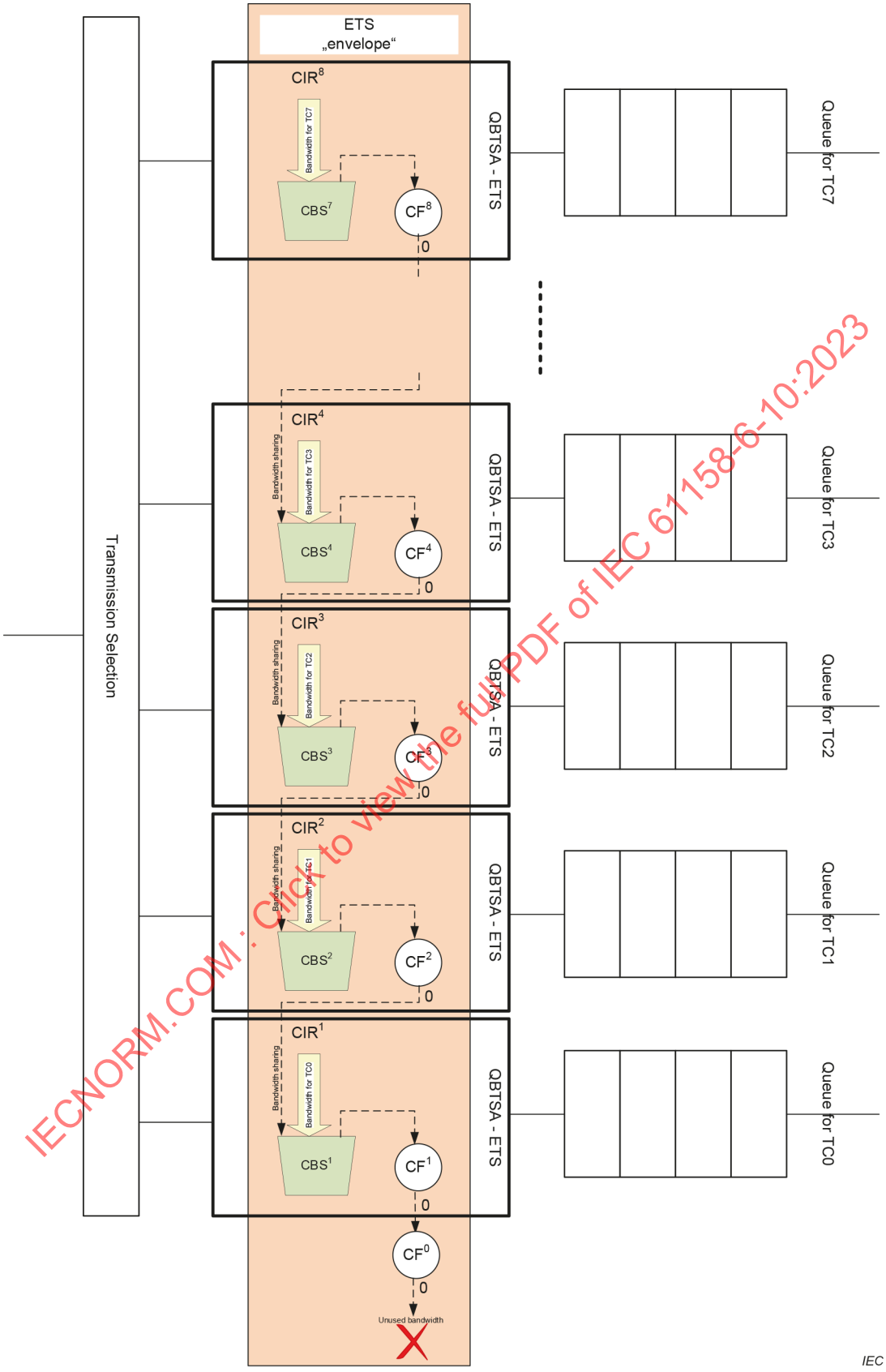


Figure 105 – Algorithm for end station ETS model

Table 381 shows the relation between the assigned managed objects and the MEF 10.3 algorithm parameters.

Table 381 – TCBandwidth

Object [IEEE Std 802.1Q]	Object [MEF 10.3]	Meaning
TCBandwidth(N)	Committed information rate (CIR)	Translates the TCBandwidth (per traffic class) value and the observation interval into CIR (per traffic class) and CBS (per traffic class) values
	Committed burst size (CBS)	
	Coupling flag (CF)	Set to 0 Enable bandwidth sharing between the selected TCs

The committed burst size per SendClock or Gating Cycle considers the number of buckets and the minimum size of these buckets. Table 382 and Table 383 show a calculation model for CBS and CIR.

Table 382 – Committed burst size

# of buckets	Minimum bucket size [Octets]	Burst size	Comment
1	1 522 (or 2 000)	1 522 (or 2 000)	If the assigned gate in one SendClock is big enough, then all frames are sent. Otherwise, a portion of the next SendClock is used for the rest. Thus, the maximum burst is larger than 25 % of a Millisecond, even if the CIR is 25 % of a Millisecond.
2	1 522 (or 2 000)	3 044 (or 4 000)	
3	1 522 (or 2 000)	4 566 (or 6 000)	
4	1 522 (or 2 000)	6 088 (or 8 000)	
…	1 522 (or 2 000)	…	
8	1 522 (or 2 000)	12 176 (or 16 000)	

Table 383 – Committed information rate

Data rate [Mbit/s]	CIR value [Mbit/s]	Comment
10	2,5	Specifies the maximum achievable information rate. The committed information rate shall be kept for an observation interval is 1 ms.
100	25	
1 000	250	
10 000	2 500	

4.12.3.3 Credit-based shaper controlled non-stream information rate

Credit-based shaper algorithm may be used as a substitution for ETS at the end station without mixing them. This algorithm with the parameters shown in Table 384 allows to assign bandwidth to the used traffic classes and thus, offers bandwidth utilization without starving the lower priorities.

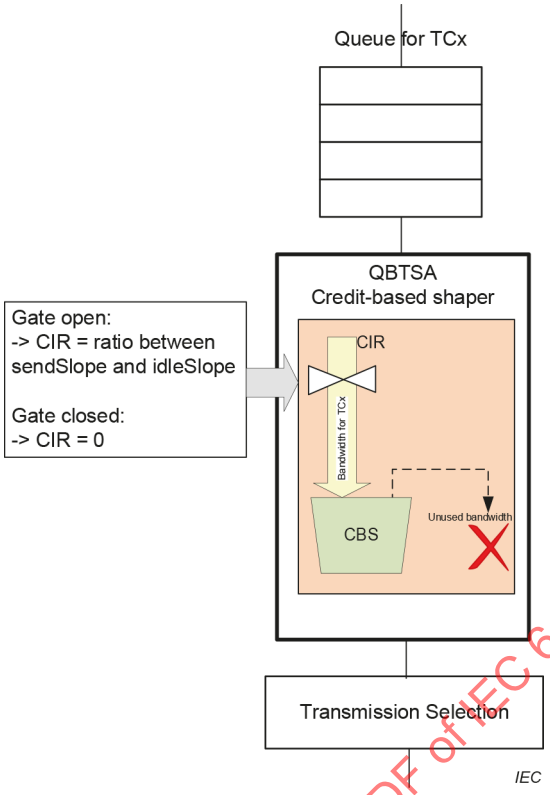


Figure 106 – Credit-based shaper algorithm

The committed burst size per SendClock or Gating Cycle considers TC3 to TC0 and the values assigned to the parameters shown in Table 384. Table 382 shows the assumed CBS and Table 383 the assumed CIR which apply according to Figure 106.

Table 384 – Credit-based shaper parameters

Object [IEEE Std 802.1Q]	Meaning
idleSlope	The rate of change of credit, in bits per second, when the value of credit is increasing (i.e., while transmit is FALSE and the transmission gate for the queue is open.)
sendSlope	The rate of change of credit, in bits per second, when the value of credit is decreasing (i.e., while transmit is TRUE).
Ratio between idleSlope and sendSlope	The rate of increase of the tokens in a token bucket. Transmitting of a frame reduces the amount of token.

4.12.3.4 Selection of managed objects for end stations

Table 385, Table 386, Table 387, and Table 388 show a selection of adapted Managed Objects according to IEEE Std 802.1Q.

DCBX (IEEE Std 802.1Q-2018, 37.2) shall not be used to configure Enhanced Transmission Selection.

Table 385 – Enhancements for scheduled traffic

Object [IEEE Std 802.1Q]	Translation	Meaning
AdminBaseTime	—	Starting point in time based on the WorkingClock and calculated according to the rules for reduction ratio, phase and sequence
AdminCycleTime	SendClockFactor	Cycle time for the periodic execution of the gate control
Gating Cycle	SendClockFactor	Cycle time for the periodic execution of the gate control
CurrentTime	WorkingClock	Timescale used for the execution of the gate control
GateControlList	SendListControl	Enables and disables the gates for selected traffic classes 0: Open TC7 to TC5 20 %: Open TC4 to TC0; Close TC7 to TC5 45 %: Close TC3 to TC0 99,9 %: Close all TCs

Table 386 – Enhanced Transmission Selection

Object [IEEE Std 802.1Q]	Value	Meaning
numTrafficClassesSupported	4	Used only for TC3, TC2, TC1 and TC0
TCPriorityAssignment(P)	TC3: yes, TC2: yes, TC1: yes, TC0:yes	All non-stream traffic classes
TCBandwidth(N)	TC3: 14 %, TC2: 5 %, TC1: 3 %, TC0: 3 %	Unused CIR from a higher class shall be able to be consumed by a lower class. The value for TCx is equal to CIR^x. Example: TC0 of 3 % of 100 Mbit/s is a CIR^0 of 3 Mbit/s Alternative: Assign the whole committed information rate (for example 25 %) to CIR^4 (used for TC3) and use only the overflow for CIR^3 to CIR^1.

Table 387 – Transmission Selection

Object [IEEE Std 802.1Q]	Value	Meaning
Transmission selection algorithm	TC7: SP, TC6: SP, TC5: SP, TC4: SP, TC3: ETS, TC2: ETS, TC1: ETS, TC0: ETS	Strict priority and Enhanced Transmission Selection shall be configured

Table 388 – Traffic classes

Object [IEEE Std 802.1Q]	Value	Meaning
Number of traffic classes (TCs)	8	Number of traffic classes supported by an end station component port.

4.12.3.5 Creating a Send List

The SendListControl covers both, non-real-time and real-time traffic as shown in Figure 107.

512 PPMs per Ethernet interface lead to up to 512 streams (supporting seamless redundancy) which are transmitted spread over phases. 512 CPMs per Ethernet interface lead to additional 512 streams (supporting seamless redundancy) transmitted by PPMs of other Ethernet interfaces in the network.

A Send List for a particular phase contains per Ethernet interface:

- for 100 Mbit/s at most 64 packets
  - HIGH / LOW / RT, or
  - RT_CLASS_3 / RT_CLASS_2 / RT_CLASS_1 / RT_CLASS_UDP.
- for 1 Gbit/s at most 256 packets
  - HIGH / LOW / RT, or
  - RT_CLASS_3 / RT_CLASS_2 / RT_CLASS_1 / RT_CLASS_UDP.

Additionally, NW, EV, CO, BEH, and BEL (non-real-time traffic) packets limited to 25 % (of the link speed) @ 1 ms by applying the selected QBTSA scheme are scheduled.

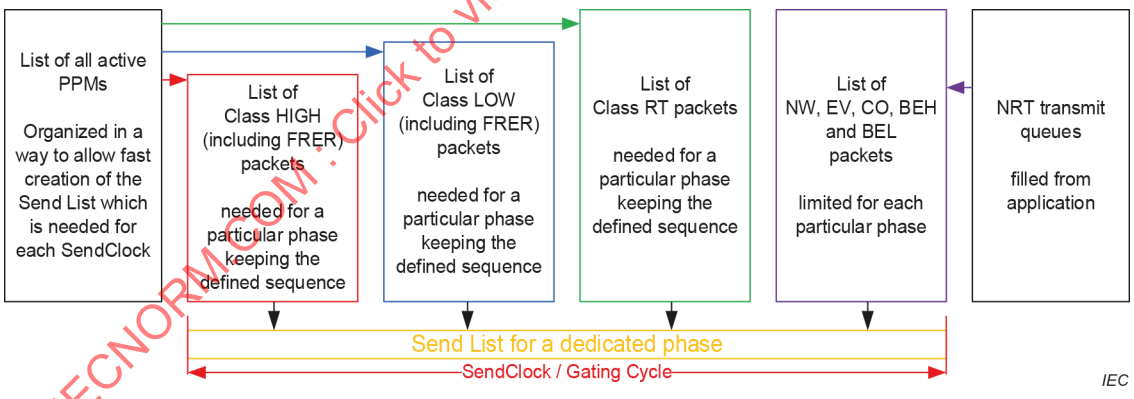


Figure 107 – Send List Feed

The content of a created Send List for a cycle varies over time due to changes in application / communication relations required from the automation process.

Table 389, Table 390 and Table 391 show the number of possible entries per SendClock / Gating Cycle into an active Send List for a dedicated phase.

Table 389 – Number of entries per SendClock per Ethernet interface at 10 Mbps

SendClock [time]	Number of entries [50 % bandwidth]
250 µs	—
500 µs	—
1 ms	With RR ≥ 8 56
2 ms	—
4 ms	—

Table 390 – Number of entries per SendClock per Ethernet interface at 100 Mbps

SendClock [time]	Number of entries [50 % bandwidth]
250 µs	17
500 µs	35
1 ms	64
2 ms	64
4 ms	64

Table 391 – Number of entries per SendClock per Ethernet interface at > 100 Mbps

SendClock [time]	Number of entries [20 % bandwidth]
31,25 µs	9
62,5 µs	17
93,75 µs	26
125 µs	35
156,25 µs	44
187,5 µs	53
218,75 µs	62
250 µs	71
500 µs	142
1 ms	256

Figure 108, Figure 109, Figure 110 and Figure 111 show the bandwidth utilization per port of a device when applying the send list defined in Figure 107 with the values from Table 391.

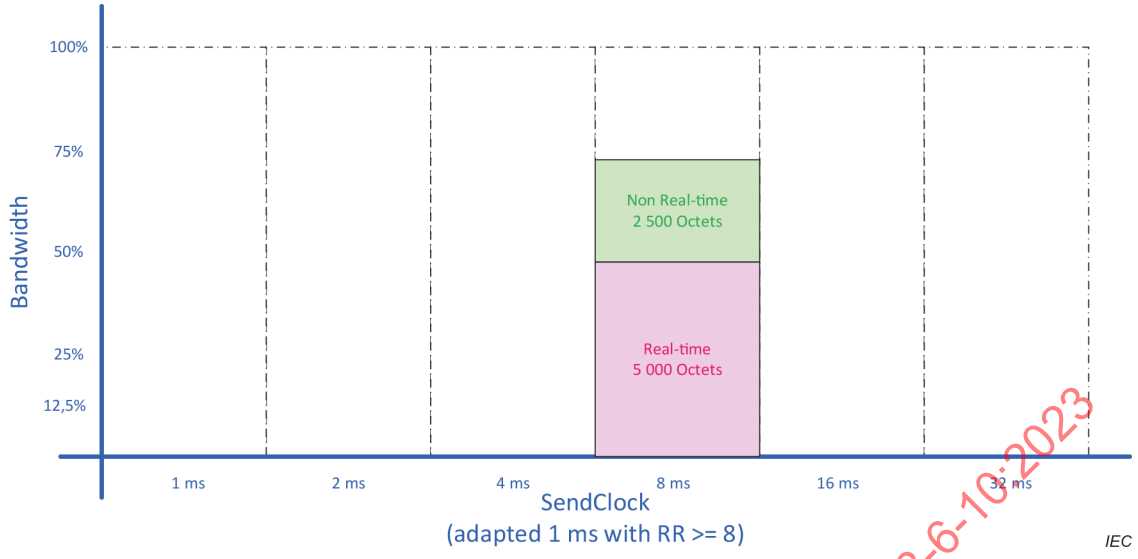


Figure 108 – Bandwidth vs. SendClock @ 10 Mbit/s

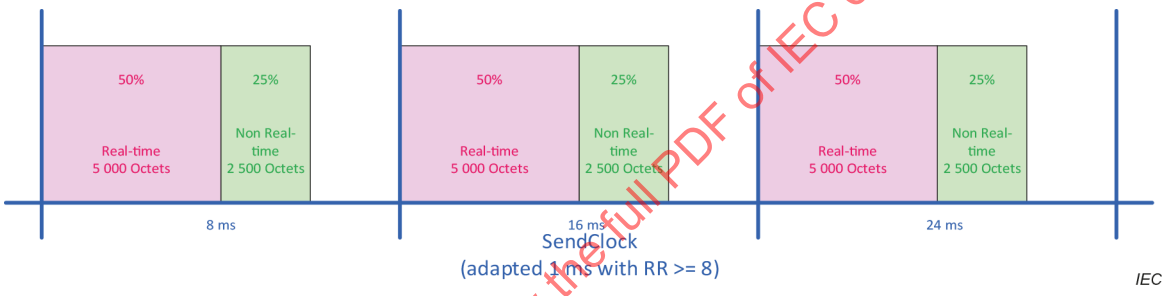


Figure 109 – 10 Mbps SendClock adaption

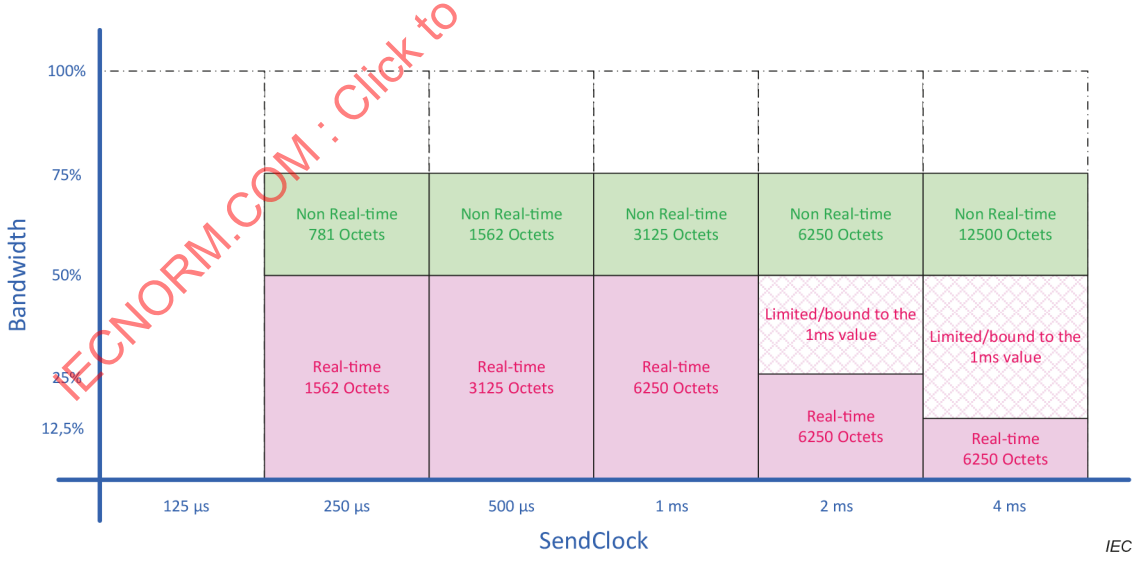


Figure 110 – Bandwidth vs. SendClock @ 100 Mbit/s

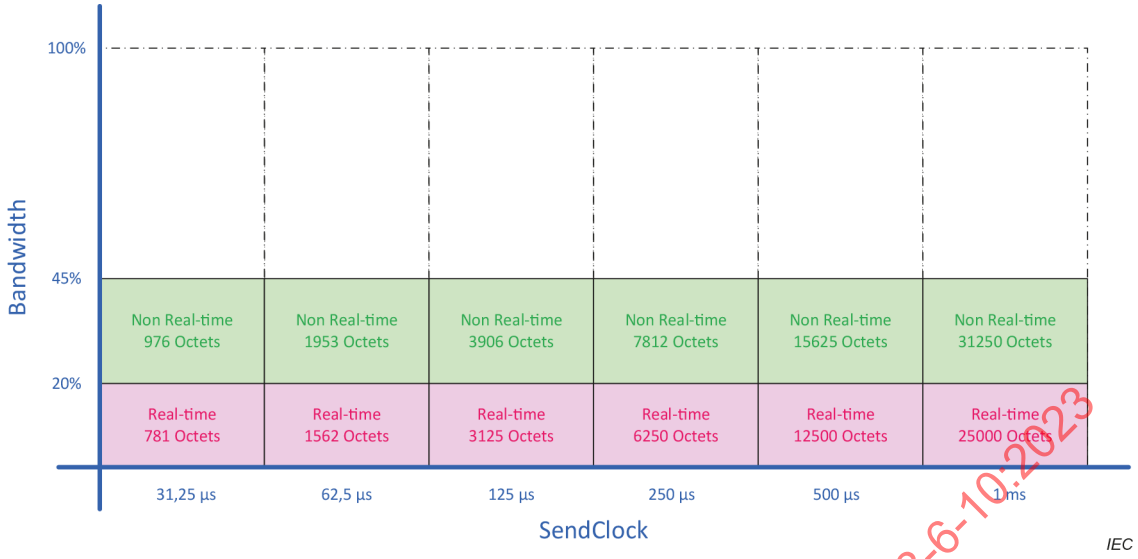


Figure 111 – Bandwidth vs. SendClock @ 1 Gbit/s

4.12.3.6 Controlling parameters

SendClockFactor, ReductionRatio, Phase and Sequence are the controlling parameters for the implementation of Network Access. Table 392 and Formula (46), Formula (47) and Formula (48) show the values.

Table 392 – SendClock and ReductionRatio

SendClock [time]	10 Mbps [Data rate]	100 Mbps [Data rate]	≥ 1 Gbps [Data rate]
31,25 µs	—	—	Together with all ReductionRatios
62,5 µs	—	—	Together with all ReductionRatios
93,75 µs	—	—	Together with all ReductionRatios
125 µs	—	—	Together with all ReductionRatios
187,5 µs	—	—	Together with all ReductionRatios
250 µs	—	Together with all ReductionRatios	Together with all ReductionRatios
375 µs	—	Together with all ReductionRatios	Together with all ReductionRatios
500 µs	—	Together with all ReductionRatios	Together with all ReductionRatios
625 µs	—	Together with all ReductionRatios	Together with all ReductionRatios
750 µs	—	Together with all ReductionRatios	Together with all ReductionRatios
875 µs	—	Together with all ReductionRatios	Together with all ReductionRatios
1 ms	Together with ReductionRatio ≥ 8	Together with all ReductionRatios	Together with all ReductionRatios
2 ms	—	Together with all ReductionRatios	—
4 ms	—	Together with all ReductionRatios	—

The ReductionRatio is created according to Formula (46).

$$\text{ReductionRatio} = 2^n \mid n \in \mathbb{N}_0 \mid n \leq 9 \quad (46)$$

where

ReductionRatio	is the result of the operation
n	is the actual factor for the operation

The Phase is created according to Formula (47).

$$\text{PhaseNumber} = 1 \text{ to ReductionRatio} \quad (47)$$

where

PhaseNumber	is the chosen one from the list
ReductionRatio	is the applied ReductionRatio

The Sequence is created according to Formula (48).

$$\text{SequenceNumber} = 1 \text{ to MaxListLength} \quad (48)$$

where

SequenceNumber	is the chosen one from the list
MaxListLength	is the maximum possible entries per Phase

4.12.3.7 Communication performance

4.12.3.7.1 Non-real-time performance

The implementation shall allow back-to-back performance for non-real-time traffic supporting the minimal IPG, for both transmit and receive direction.

4.12.3.7.2 Real-time performance

The implementation shall allow back-to-back performance for real-time traffic supporting the minimal IPG, for both transmit and receive direction.

4.12.3.8 End station FRER

See IEEE Std 802.1CB

Frame replication and elimination in the end stations allows the optimization of the transmission by SendListControl.

Frames are replicated by using one or two PPMs (same FrameID, different TCI.VID) and eliminated by using one CPM.

Thus, the defined optimizations for streams apply for each of the replicated frames.

4.12.3.9 Stream queues

IEEE Std 802.1Q-2018, Figure 34-1 defines the concept of Talker (or stream) based queues which provide means to implement reduction ratio, phase, sequence and offset.

This entity is name time-aware offset control and used to implement the SendListControl defined in this document.

4.12.3.10 Queues

IEEE Std 802.1Q defines the concept of traffic classes based on queues.

Network Control frames, e.g., IEEE Std 802.1AS or IEEE Std 802.1AB, are defined without a VLAN tag. These frames are queued into network control by identifying them as network control traffic.

End station components shall support eight queues according to:

– Table 393 for time-aware end stations without time-aware streams,
– Table 395 for time-aware end stations with time-aware streams,
– Table 397 for non-time-aware end stations without RT_CLASS_3, and
– Table 399 for non-time-aware end stations with RT_CLASS_3.

Frames without VLAN tag in non-time-aware systems, which are not network control protocol, are assigned to the BEL.

This is one building block to implement the SendListControl defined in this document.

4.12.3.11 Queue based transmission selection algorithm

IEEE Std 802.1Q defines the concept of Queue based transmission selection algorithm.

End station components shall support queue based transmission selection algorithms according to:

– Table 393 for time-aware end stations without time-aware streams,
– Table 395 for time-aware end stations with time-aware streams,
– Table 397 for non-time-aware end stations without RT_CLASS_3, and
– Table 399 for non-time-aware end stations with RT_CLASS_3.

With the values according to Table 386.

This is one building block to implement the SendListControl defined in this document.

4.12.3.12 Queue masking for end stations

4.12.3.12.1 General

IEEE Std 802.1Q defines the concept of queue masking / enhancements for scheduled traffic for end station components.

This is one building block to implement the SendListControl defined in this document.

4.12.3.12.2 Time-aware end station – without time-aware streams

End station components shall support queue masking according to:

– Table 393, Table 394 and Figure 112 for time-aware end stations without time-aware streams.

Table 393 – Queue usage – time-aware end station – without time-aware streams

Queue	TCI.PCP	Stream	Preemption [a]	Mask [Group]	QBTSA	Comment
0	0	—	Preemptable	Other	SP, ETS	BEL Inter region traffic, passing by. Other protocols
1	1	—	Preemptable	Other	SP, ETS	BEH Inter region traffic, passing by. Other protocols
2	2	—	Preemptable	Other	SP, ETS	CO Protocols used for connection establishment, records and control Domain internal traffic
3	3	—	Preemptable	Other	SP, ETS	EV Non-stream acyclic transmitted frames RTA_CLASS_X
4	7	—	Preemptable	Other	SP	NW Network control [b] e.g. synchronization, neighborhood
5	4	RT	Express	Stream	SP	RT Cyclic transmitted frames rated by this document as “Streams, class RT” Bandwidth control, learned path
6	5	LOW	Express	Stream	SP	LOW Unused Transmission blocked by queue bases gate control
7	6	HIGH	Express	Stream	SP	HIGH Unused Transmission blocked by queue bases gate control

[a] Activate if supported.

[b] Network control protocols, for example LLDP, gPTP or RSTP, mostly not use VLANs. Their local protocol agent enqueues these frames directly to the appropriate queue.

SP: Strict Priority

ETS: Enhanced Transmission Selection

QBTSA: Queue Based Transmission Selection Algorithm

The values in the column Time in Table 394 are just examples. The number of lines represents the number of Gate Control List entries needed to apply this setup.

Table 394 – Queue masking – time-aware end station – without time-aware streams

Time [Link speed]	Queue 7	Queue 6	Queue 5	Queue 4	Queue 3	Queue 2	Queue 1	Queue 0	Comment
0	Closed	Closed	Open	Closed	Closed	Closed	Closed	Closed	Only streams
20 %	Closed	Closed	Closed	Open	Open	Open	Open	Open	Only non-stream
45 %	Closed	Closed	Closed	Open	Closed	Closed	Closed	Closed	Only network control
99 %	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Guard band

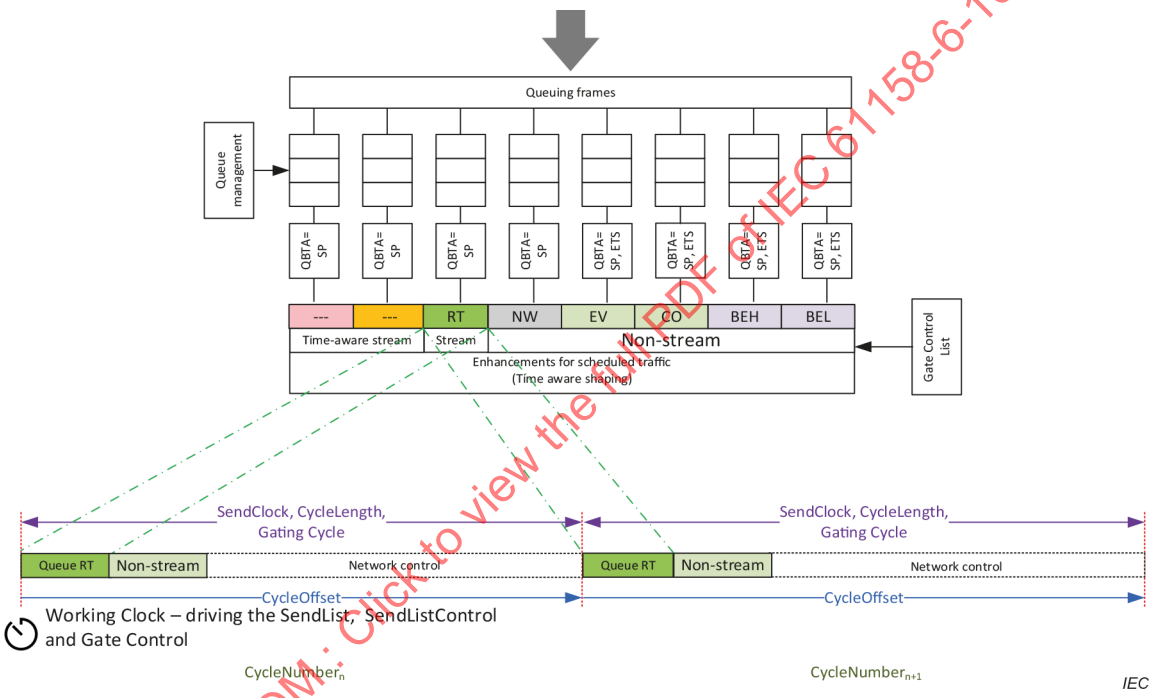


Figure 112 – Queue masking – time-aware end stations – without time-aware streams

4.12.3.12.3 Time-aware end station – with time-aware streams

End station components shall support queue masking according to:

– Table 395, Table 396 and Figure 113 for time-aware end stations with time-aware streams.

Table 395 – Queue usage – time-aware end station – with time-aware streams

Queue	TCI.PCP	Stream	Preemption	Mask [Group]	QBTSA	Comment
0	0	—	Preemptable	Other	SP, ETS	BEL Inter region traffic, passing by. Other protocols
1	1	—	Preemptable	Other	SP, ETS	BEH Inter region traffic, passing by. Other protocols
2	2	—	Preemptable	Other	SP, ETS	CO Protocols used for connection establishment, records and control Domain internal traffic
3	3	—	Preemptable	Other	SP, ETS	EV Non-stream acyclic transmitted frames RTA_CLASS_X
4	7	—	Preemptable	Other	SP	NW Network control [a] e.g. synchronization, neighborhood
5	4	RT	Express	Stream	SP	RT Cyclic transmitted frames rated by this document as “Streams, class RT” Bandwidth control, learned path
6	5	LOW	Express	Stream	SP	LOW Cyclic transmitted frames rated by this document as “Time-aware streams, class LOW” Zero congestion loss, calculated path
7	6	HIGH	Express	Stream	SP	HIGH Cyclic transmitted frames rated by this document as “Time-aware streams, class HIGH” Deadline, zero congestion loss, calculated path

[a] Network control protocols, for example LLDP, gPTP or RSTP, mostly do not use VLANs. Their local protocol agent enqueues these frames directly to the appropriate queue.

SP: Strict Priority

ETS: Enhanced Transmission Selection

QBTSA: Queue Based Transmission Selection Algorithm

The values in the column Time in Table 396 are just examples. The number of lines represents the number of Gate Control List entries needed to apply this setup.

Table 396 – Queue masking – time-aware end station – with time-aware streams

Time [Link speed]	Queue 7	Queue 6	Queue 5	Queue 4	Queue 3	Queue 2	Queue 1	Queue 0	Comment
0	Open	Open	Open	Closed	Closed	Closed	Closed	Closed	Only streams
20 %	Closed	Closed	Closed	Open	Open	Open	Open	Open	Only non-stream
45 %	Closed	Closed	Closed	Open	Closed	Closed	Closed	Closed	Only network control
99 %	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Guard band

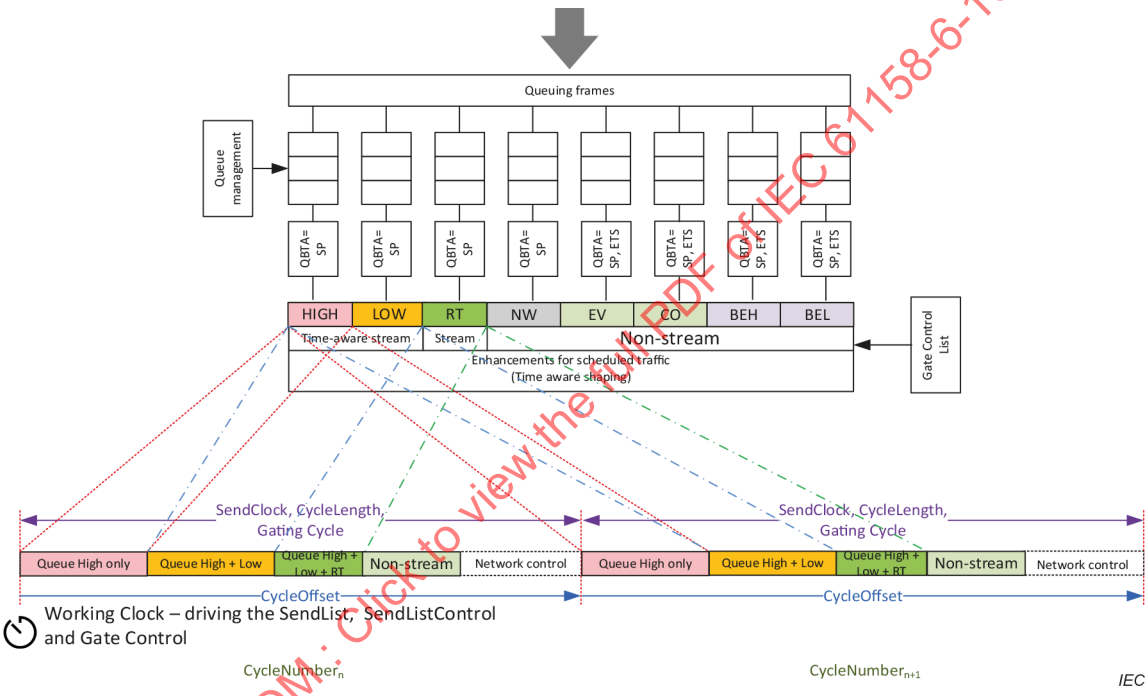


Figure 113 – Queue masking – time-aware end station – with time-aware streams

4.12.3.12.4 Non-time-aware end station – without RT_CLASS_3

End station components shall support queue masking according to:

– Table 397, Table 398 and Figure 114 for non-time-aware without RT_CLASS_3 end stations.

Table 397 – Queue usage – non-time-aware end station – without RT_CLASS_3

Queue	TCI.PCP	Stream	Preemption	Mask [Group]	QBTSA	Comment
0	0	—	Preemptable	Other	SP, ETS	BEL Inter region traffic, passing by. Other protocols
1	1, 3 and 4	—	Preemptable	Other	SP, ETS	BEH Inter region traffic, passing by. Other protocols
2	2	—	Preemptable	Other	SP, ETS	CO Protocols used for connection establishment, records and control Domain internal traffic
3	5	—	Preemptable	Other	SP, ETS	EV Non-stream acyclic transmitted frames RTA_CLASS_X
4	7	—	Preemptable	Other	SP	NW Network control [a] e.g. synchronization, neighborhood
5	6	RT	Express	Stream	SP	RT Cyclic transmitted frames rated by this document as RT_CLASS_X Bandwidth control, learned path
6	—	—	Preemptable	Other	SP	LOW Unused Transmission blocked by queue bases gate control
7	—	—	Preemptable	Other	SP	HIGH Unused Transmission blocked by queue bases gate control

[a] Network control protocols, for example LLDP, gPTP or RSTP, mostly do not use VLANs. Their local protocol agent enqueues these frames directly to the appropriate queue.

SP: Strict Priority

ETS: Enhanced Transmission Selection

QBTSA: Queue Based Transmission Selection Algorithm

The values in the column Time in Table 398 are just examples. The number of lines represents the number of Gate Control List entries needed to apply this setup.

Table 398 – Queue masking – non-time-aware end station – without RT_CLASS_3

Time [Link speed]	Queue 7	Queue 6	Queue 5	Queue 4	Queue 3	Queue 2	Queue 1	Queue 0	Comment
0	Closed	Closed	Open	Closed	Closed	Closed	Closed	Closed	Only streams
20 %	Closed	Closed	Closed	Open	Open	Open	Open	Open	Only non-stream
45 %	Closed	Closed	Closed	Open	Closed	Closed	Closed	Closed	Only network control
99 %	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Guard band

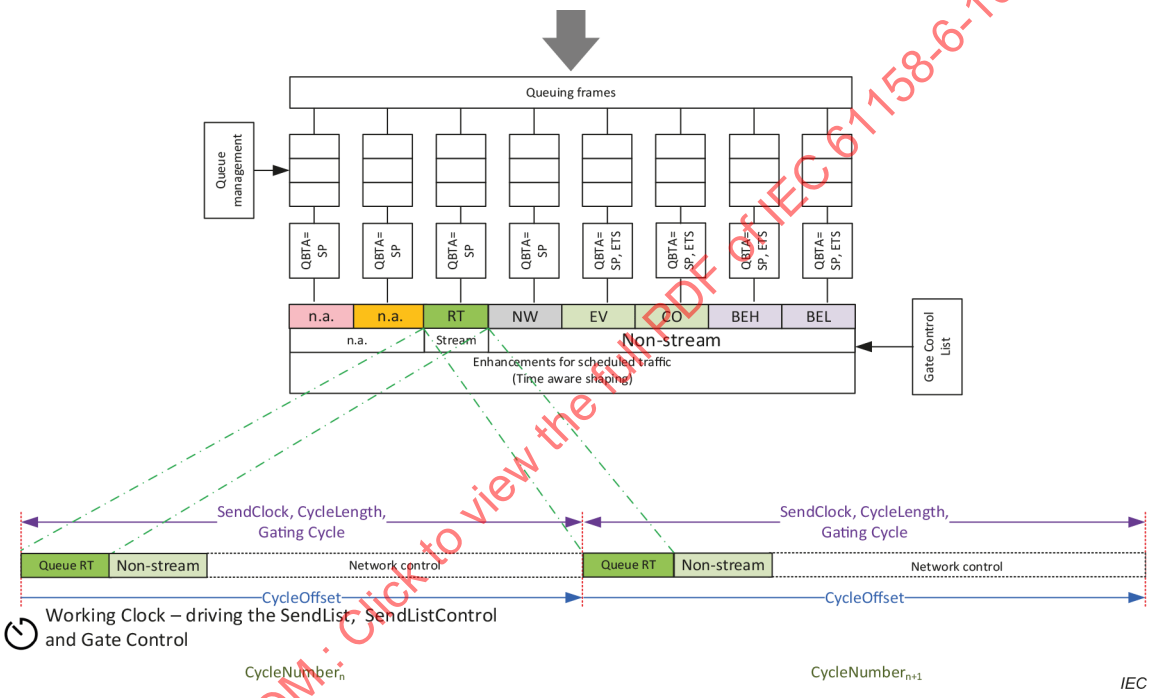


Figure 114 – Queue masking – non-time-aware – without RT_CLASS_3

4.12.3.12.5 Non-time-aware end station – with RT_CLASS_3

End station components shall support queue masking according to:

– Table 399, Table 400 and Figure 115 for non-time-aware with RT_CLASS_3 end stations.

Table 399 – Queue usage – non-time-aware end station – with RT_CLASS_3

Queue	TCI.PCP	Stream	Preemption	Mask [Group]	QBTSA	Comment
0	0	—	Preemptable	Other	SP, ETS	BEL Inter region traffic, passing by. Other protocols
1	1, 3 and 4	—	Preemptable	Other	SP, ETS	BEH Inter region traffic, passing by. Other protocols
2	2	—	Preemptable	Other	SP, ETS	CO Protocols used for connection establishment, records and control Domain internal traffic
3	5	—	Preemptable	Other	SP, ETS	EV Non-stream acyclic transmitted frames RTA_CLASS_X
4	7	—	Preemptable	Other	SP	NW Network control [a] e.g. synchronization, neighborhood
5	6	RT	Express	Stream	SP	RT Cyclic transmitted frames rated by this document as RT_CLASS_X Bandwidth control, learned path
6	—	—	Preemptable	Other	SP	LOW Unused Transmission blocked by queue bases gate control
7	n.a.	—	Express, Fragmentation	RED	SP	HIGH RT_CLASS_3 frames are enqueued into this queue. The time-aware control is used to transmit the frame according to its FrameSendOffset.

[a] Network control protocols, for example LLDP, gPTP or RSTP, mostly do not use VLANs. Their local protocol agent enqueues these frames directly to the appropriate queue.

SP: Strict Priority

ETS: Enhanced Transmission Selection

QBTSA: Queue Based Transmission Selection Algorithm

Fragmentation: Support fragmentation according to this document

The values in the column Time in Table 400 are just examples. The number of lines represents the number of Gate Control List entries needed to apply this setup.

Table 400 – Queue masking – non-time-aware end station – with RT_CLASS_3

Time [Link speed]	Queue 7	Queue 6	Queue 5	Queue 4	Queue 3	Queue 2	Queue 1	Queue 0	Comment
0	Open	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Only RED
EndOfRED	Open	Closed	Open	Closed	Closed	Closed	Closed	Closed	Only stream
EndOfRED + 10 %	Closed	Closed	Open	Open	Open	Open	Open	Open	Stream and non-stream
45 %	Closed	Closed	Closed	Open	Closed	Closed	Closed	Closed	Network control
99 %	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Closed	Guard band

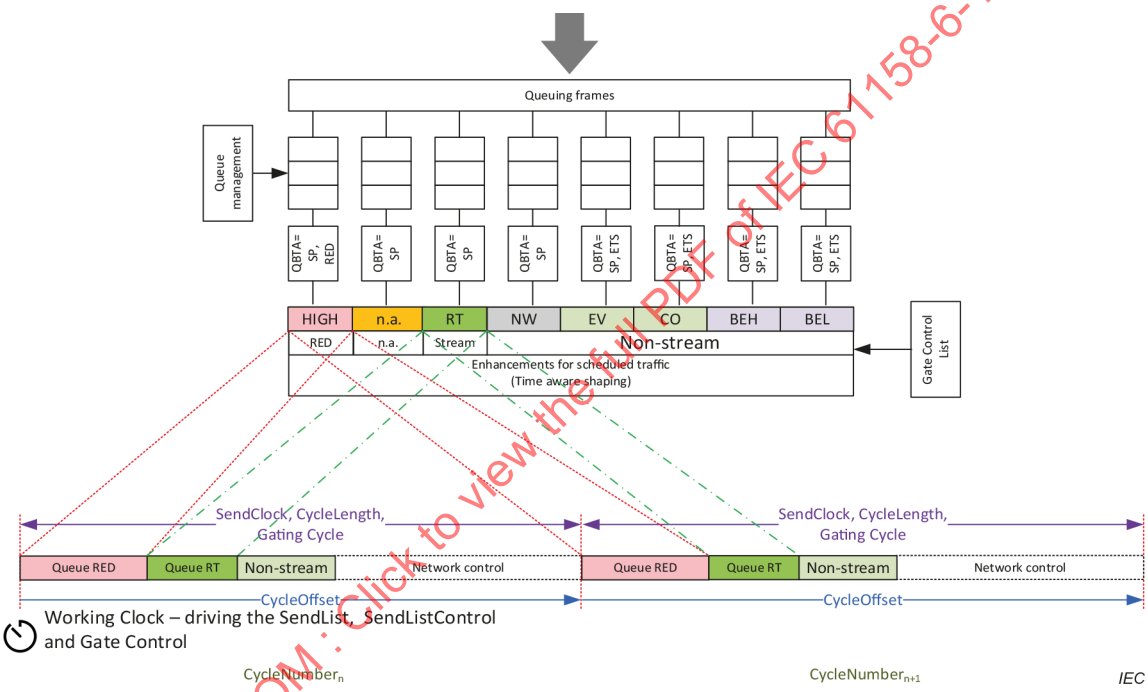


Figure 115 – Queue masking – non-time-aware end station – with RT_CLASS_3

4.12.3.13 Pre-emption

IEEE Std 802.1Q and IEEE Std 802.3 define the concept of pre-emption for end stations.

End station components shall support configuring pre-emption on a per queue base.

4.12.3.14 Examples

Figure 116 and Figure 117 show examples for end stations.

Figure 116 shows an end station system with one integrated end station component.

Figure 117 shows an end station system which contain multiple end station components.

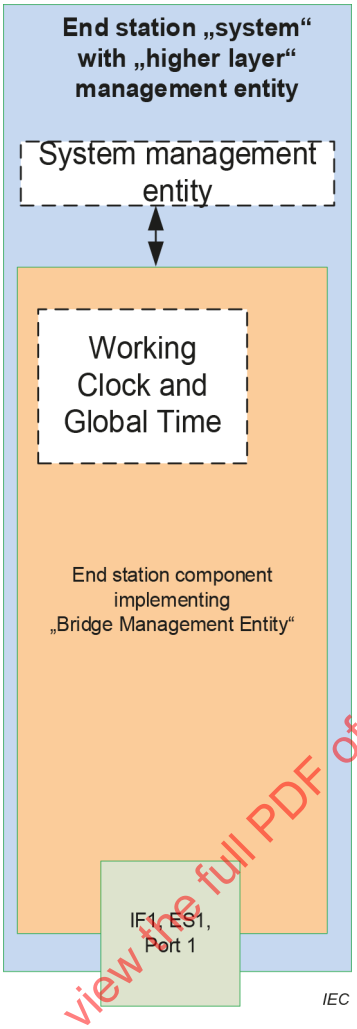


Figure 116 – End station

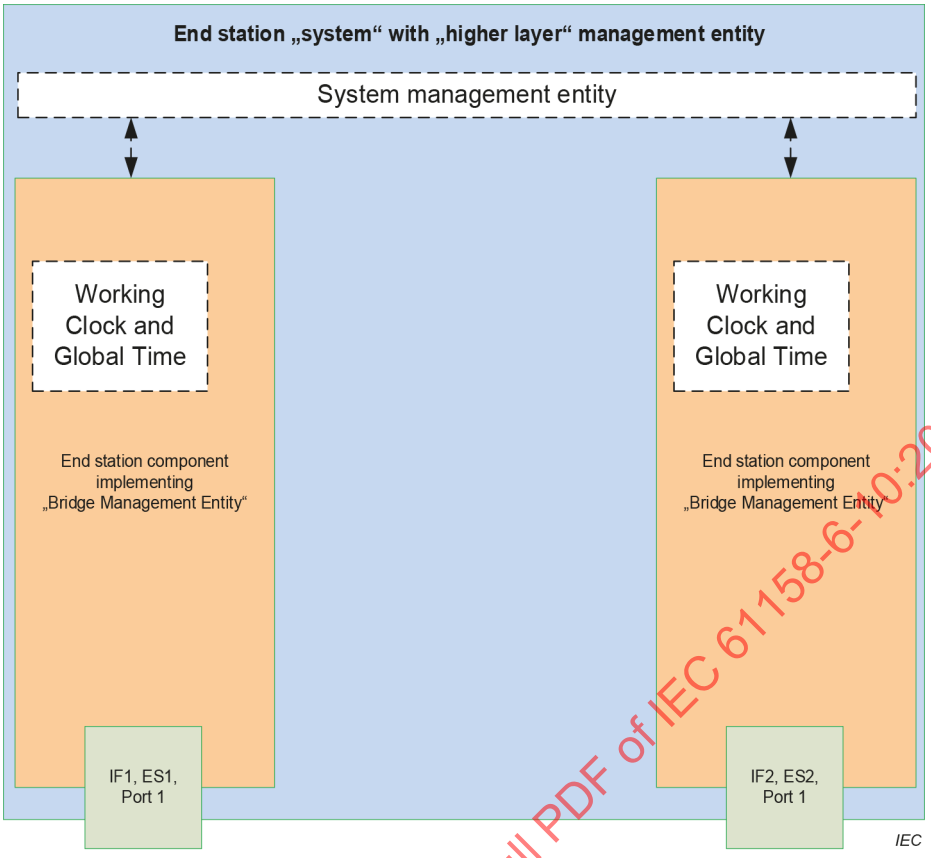


Figure 117 – End station System – with multiple end station components

4.12.4 Bridge

4.12.4.1 General

Subclause 4.12.4 selects features out of:

– IEEE Std 802.1Q,
– IEEE Std 802.1CB, and
– IEEE Std 802.3.

The implementation and usage of more than this selection is in the responsibility of the device manufacturer.

Figure 118 and Figure 140 show a bridge following the IEEE Std 802.1Q-2018, Figure 8-8 and IEEE Std 802.1Q-2018, 8.3, NOTE 3 specified model.

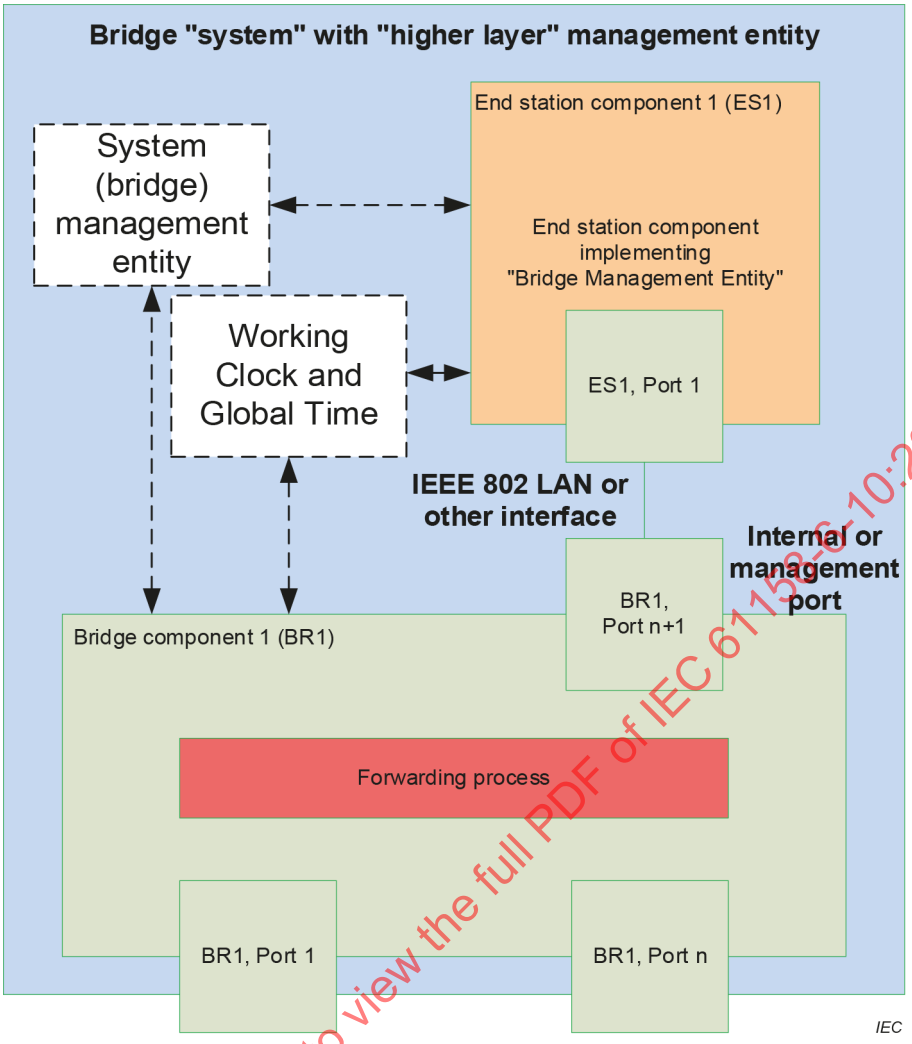


Figure 118 – System incorporating a bridge

4.12.4.2 Bridge Stream identification and translation

See IEEE Std 802.1CB-2017, Clause 6.

Figure 122 shows the setup of boundary ports in case of inter NME domain streams. In this case, the ingress port shall support “Active Destination MAC and VLAN Stream identification” according to IEEE Std 802.1CB-2017, 6.6 for the stream translation between the different NME domains.

4.12.4.3 Bridge Port Transmit and Receive

4.12.4.3.1 General

See IEEE Std 802.1Q-2018, 8.5 and IEEE Std 802.1Q-2018, 6.9.

If a port is defined as Boundary Port for a NME domain, then this module executes tagging and untagging. Table 401 and Table 402 show a selection of required managed objects.

These frames are tagged on receive and untagged on transmit at a Boundary Port.

A Boundary Port accepts on receive only:

– Untagged and Priority-tagged frames

A Boundary Port creates on transmit only:

– Untagged or Priority-tagged frames

NOTE A bridge can support forwarding of priority tagged frames.

Table 401 – Selection of managed objects for ingress

Managed object	Comment
Acceptable Frame Types	Selects the frame type which can “pass” the ingress rule
Enable Ingress Filtering	Enables or disables the “Acceptable Frame Types” setting
Port VLAN Identifier (PVID)	Defines the C-VLAN used for untagged, or priority tagged frames
Default ingress priority	Defines the PCP value used together with the C-VLAN for untagged frames
VID translation table	The VID translation table provides a mapping between “local VID” values and “relay VID” values

Table 402 – Selection of managed objects for egress

Managed object	Comment
Static VLAN Registration Entry VLAN	Defines the port as member of an untagged member set
Egress VID translation table	The Egress VID translation table provides a mapping between “local VID” values and “relay VID” values

Figure 119 shows the domain protection concept of time-aware bridges which is implemented by performing a TCI.PCP remapping / TCI.VID assignment at the boundary port. Each port of an Ethernet Bridge may be used as boundary port. The TCI.PCP remapping / TCI.VID assignment should not influence the bridge delay.

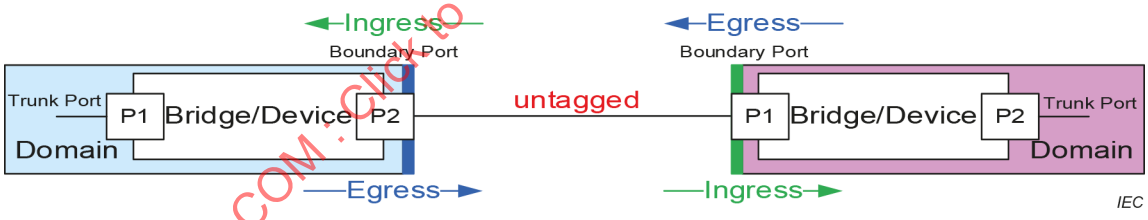


Figure 119 – Domain Boundary

4.12.4.3.2 Time-aware bridge

Ports are either Boundary Ports or Trunk Ports.

End stations are either connected to a Trunk Port (in this case their Ethernet interface is part of the NME domain) or to a Boundary Port (in this case their Ethernet interface is NOT part of the NME domain).

A Trunk Port:

– Forwards and receives only VLAN-tagged frames

Ports which are used for inter NME domain streams are Trunk Ports with a dedicated setup.

4.12.4.3.3 Non-time-aware bridge

Used as specified by IEEE Std 802.1Q.

4.12.4.4 Priority Regeneration Override

4.12.4.4.1 General

See IEEE Std 802.1Q-2018, 12.20.3

If a port is defined as Boundary Port for a domain, then this module executes Priority Regeneration for ingress traffic. Additionally, the frame will be tagged with the VID of the domain, if untagged or priority tagged. An ingress port shall support “Priority regeneration”.

Peer to peer protocols like IEEE Std 802.1AS or IEEE Std 802.1AB are not affected by this module.

4.12.4.4.2 Time-aware bridge

4.12.4.4.2.1 General

Check LLDP TLVs (see 4.12.6) whether the neighbor is part of the own NME domain or is a non-time-aware device according to this document or is part of another NME domain.

Depending on the neighbor, the port is setup as:

- Trunk Port or
- Boundary Port
  - to a non-time-aware device according to this document
  - to any other station
  - to another NME domain, with and without inter NME domain traffic

4.12.4.4.2.2 Port is domain boundary port supporting queue resources for RT_CLASS_STREAM, class RT

Figure 120 shows the setup of boundary ports in case of neighboring non-time-aware devices according to this document.

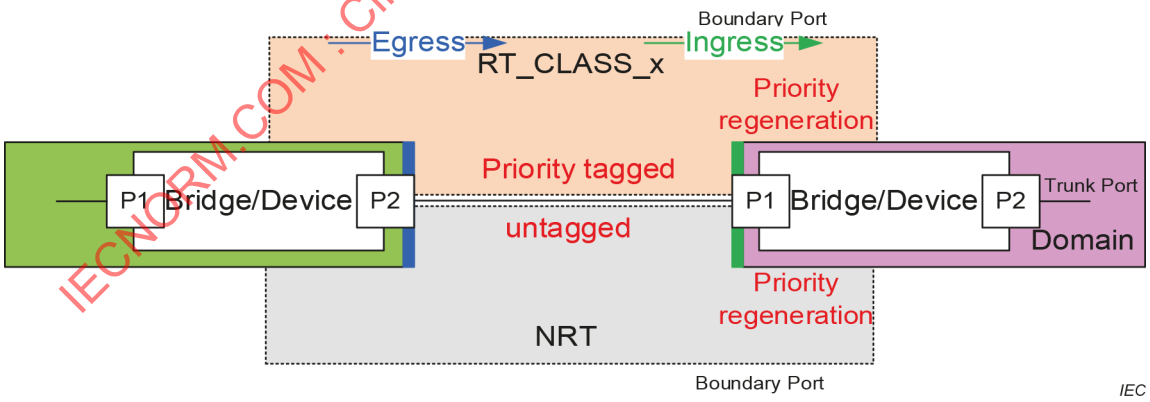


Figure 120 – Domain Boundary – RT_CLASS_STREAM, class RT

The priority of a received frame will be overridden according to Table 403.

Table 403 – Priority remapping at an ingress boundary port connected to a non-time-aware device according to this document

TCI.PCP Source (outside NME domain)	->	TCI.PCP Destination (inside NME domain)
7	is overridden to	1 (BEH)
6 (assumed RT)		4 (RT)
5 (assumed RTA)		3 (EV)
4		1 (BEH)
3		0 (BEL)
2 (assumed RPC)		0 (BEL)
1		0 (BEL)
0 (assumed DCP)		0 (BEL)
Untagged		0 (BEL)

4.12.4.4.2.3 Port is domain boundary port

Figure 121 shows the setup of boundary ports in case of unknown neighbors.

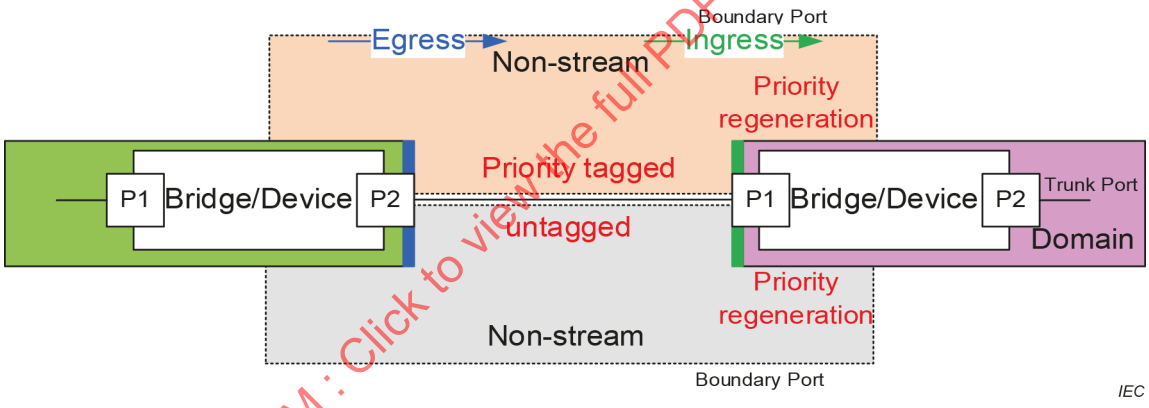


Figure 121 – Domain Boundary – Boundary Port

The priority of a received frame will be overridden according to Table 404.

Table 404 – Priority remapping at a domain ingress boundary port

TCI.PCP Source (outside NME domain)	->	TCI.PCP Destination (inside NME domain)
7	is overridden to	1 (BEH)
6		1 (BEH)
5		1 (BEH)
4		1 (BEH)
3		0 (BEL)
2		0 (BEL)
1		0 (BEL)
0		0 (BEL)
Untagged		0 (BEL)

4.12.4.4.2.4 Port is inter NME domain boundary port

Figure 122 shows the setup of boundary ports in case of inter NME domain streams. In this case, the ingress port shall support “Active Destination MAC and VLAN Stream identification” according to IEEE Std 802.1CB-2017, 6.6 for the stream translation between the different NME domains.

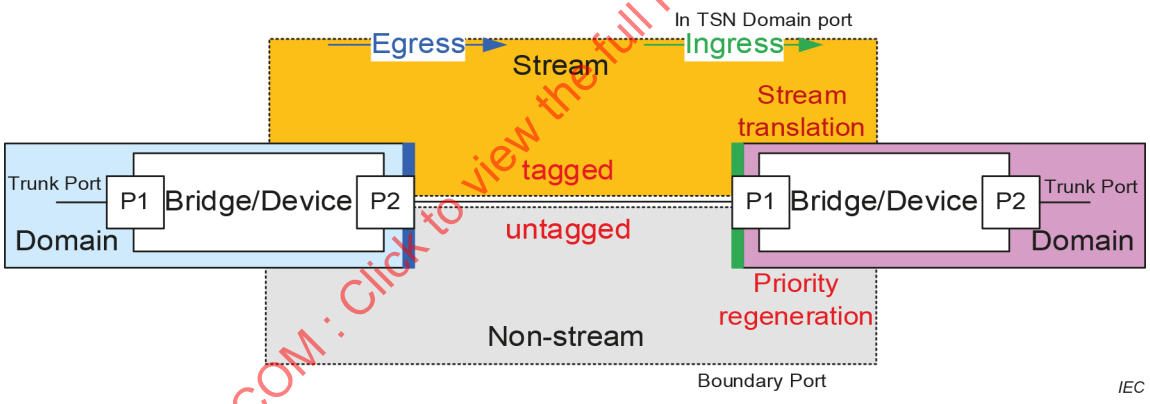


Figure 122 – Domain Boundary – Inter NME domain streams

The egress port handles in this case the stream and the non-stream VLANs (according to Table 414) differently. Frames from non-stream VLANs are untagged, while frames from stream VLANs are unchanged forwarded according the FDB entry.

Untagged or priority tagged frames (non-stream VLANs) will be overridden according to Table 405.

Table 405 – Priority remapping at a domain ingress boundary port

TCI.PCP Source (outside NME domain)	->	TCI.PCP Destination (inside NME domain)
7	is overridden to	1 (BEH)
6		1 (BEH)
5		1 (BEH)
4		1 (BEH)
3		0 (BEL)
2		0 (BEL)
1		0 (BEL)
0		0 (BEL)
Untagged		0 (BEL)

Stream VLAN tagged frames will be overridden according to Table 406. If a received frame (stream) is not found in the stream translation table, then it shall be dropped.

Table 406 – "Active Destination MAC and VLAN Stream identification" at a domain ingress boundary port

Source Destination MAC address TCI.VID	->	Destination Destination MAC address TCI.VID TCI.PCP
Compare value of Destination MAC address "A" and TCI.VID "B"	is overridden to	Replace it with Destination MAC address "C", TCI.VID "D" and TCI.PCP "E"
…	is overridden to	…
Compare value of Destination MAC address "Y" and TCI.VID "Z"	is overridden to	Replace it with Destination MAC address "G", TCI.VID "H" and TCI.PCP "I"

4.12.4.4.3 Non-time-aware bridge

Used as specified by IEEE Std 802.1Q.

4.12.4.5 Reception Port State

See IEEE Std 802.1Q-2018, 8.13.9 and IEEE Std 802.1AC.

Two flags are used for port control:

– MAC_Operational (See IEEE Std 802.1AC)
– Administrative Bridge Port State

If MAC_Operational is set to FALSE or Administrative Bridge Port State is set to Disabled then both forwarding and learning are disabled for all trees.

Each tree, for example defined by

– IEC 62439-2 or
– IEEE Std 802.1AS or
– IEEE Std 802.1AB or
– VID,

defines its own tree-related reception port state.

4.12.4.6 Transmission Port State

See IEEE Std 802.1Q-2018, 8.13.9 and IEEE Std 802.1AC.

Two flags are used for port control:

– MAC_Operational (See IEEE Std 802.1AC)
– Administrative Bridge Port State

If MAC_Operational is set to FALSE or Administrative Bridge Port State is set to Disabled then both forwarding and learning are disabled for all trees.

Each tree, for example defined by

– IEC 62439-2 or
– IEEE Std 802.1AS or
– IEEE Std 802.1AB or
– VID,

defines its own tree-related reception port state.

4.12.4.7 Filtering Database

4.12.4.7.1 General

The general definition is valid for time-aware and non-time-aware bridges and following the Draft IEC/IEEE 60802.

According to IEEE Std 802 and this document a switch contains a Filtering Data Base (FDB) to optimize the bridging.

The FDB supports learning of source MAC address, either IVL and/or SVL, queries for destination MAC address and VID, and management of static entries (local interface or protocol based) for the supported trees.

The expected number of entries results from the maximum number of stations of 1 024 together with the 50 % utilization of the entries due to hash usage.

For non-stream FDBs, Table 407 shows the expected number of possible entries, Table 408 the expected neighborhood for hash based FDBs, and Table 409 the FDB attributes.

Table 407 – Number of FDB entries

# of VLANs	# of entries	Usage
1	2 048	Learned and static entries for both, Unicast and Multicast. Additional VLANs may be supported

Table 408 – Neighborhood for hashed entries

Neighborhood [a]	Usage
4	Default A neighborhood of four entries is used to store a learned entry if the hashed entry is already used.
8	Recommended A neighborhood of eight entries is used to store a learned entry if the hashed entry is already used.
[a] A neighborhood of x adjacent entries for the hashed index is checked when adding, seeking, or updating an entry in the FDB.	

Table 409 – FDB attributes for "Non streams"

Flag	Value	Usage
Learning	Yes	Default Learn the receive SA-MAC addresses
Default Forwarding Rule	Flood	Default Flood on all ports if no explicit forwarding rule is stated
Shared VLAN Learning	No	Default Each VLAN learns individually

Figure 123 shows the basic handling model of IEEE Std 802.1AC and IEEE Std 802.1Q for LLC protocols.

This model ensures that these protocols are:

– not learned,

– not influence by any active topology, and

– not ingress or egress filtered.

Table 410 shows the MAC addresses which are bound to protocols which define their own active topology independently from the active topology established by MRP or RSTP.

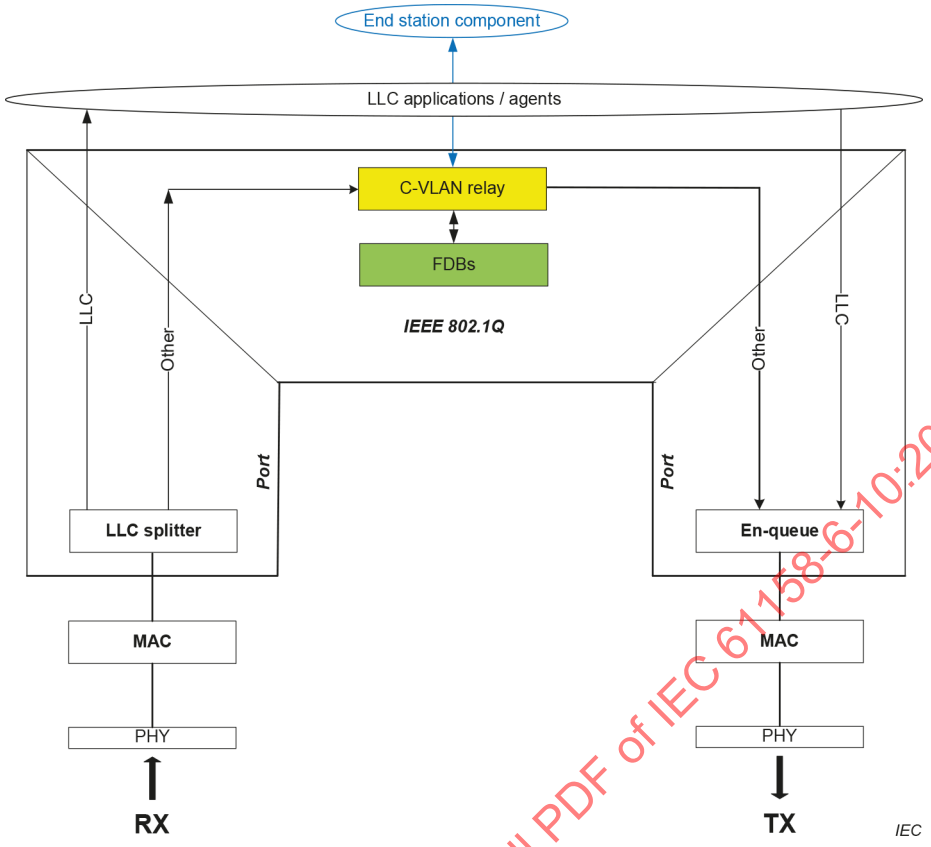


Figure 123 – LLC protocol flow

Protocols which are directly forwarded to their LLC based agent, for example IEEE Std 802.1AS or IEEE Std 802.1AB, are never learned by an FDB, because the frames are not seen – by IEEE Std 802.1AC definition – by the FDB based forwarding process.

Table 410 – List of MAC address

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Application	Meaning
01-0E-CF	00-04-20	Receive [a]	In conjunction with PTCP-RTSyncPDU with follow up and FrameID(=0x0020) used for working clock synchronization
01-0E-CF	00-04-40	Receive [a]	In conjunction with PTCP-FollowUpPDU and FrameID(=0xFF20) used for working clock synchronization
01-0E-CF	00-04-80	Receive [a]	In conjunction with PTCP-RTSyncPDU and FrameID(=0x0080) used for working clock synchronization
01-0E-CF	00-05-00	Receive [a]	Additional managed multicast MAC address used vendor specific in conjunction with MRP.
01-80-C2	00-00-00	Receive [a]	IEEE Std 802.1Q Rapid spanning tree protocol (RSTP)
01-80-C2	00-00-01 – 00-00-0E	Receive [a]	IEEE Std 802.1Q
01-80-C2	00-00-0E	Receive [a]	IEEE Std 802.1AB (LLDP)
01-80-C2	00-00-0E	Receive [a]	IEEE Std 802.1AS (gPTP)

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Application	Meaning
01-80-C2	00-00-0E	Receive [a]	PTCP In conjunction with PTCP-DelayReqPDU and FrameID(=0xFF40), PTCP-DelayResPDU with follow up and FrameID(=0xFF41), PTCP-DelayFuResPDU and FrameID(=0xFF42) and PTCP-DelayResPDU without follow up and FrameID (=0xFF43) used for delay measurement
01-80-C2	00-00-0F	Receive [a]	IEEE Std 802.1Q
01-80-C2	00-00-10	Receive [a]	IEEE Std 802.1Q All LANs bridge management group address
Others	Others	—	—

[a] If applicable.

4.12.4.7.2 Basic FDB entries

Table 411, Table 412 and Table 413 show the list of entries.

Each active topology, either defined by VLAN or the used protocol, defines its own forwarding rules.

The entry “No” refers to the active topology established by MRP or RSTP or other means.

NOTE IEC 62439-2, if supported, adds further entries in the FDB.

Learned FDB entries of a port shall be deleted when a Link Down is detected. Flushing all learned entries, as needed by MRP or RSTP, shall be supported.

Unknown (not found in the FDB) Destination MAC addresses are handled according to the default forwarding rule as shown in Table 414.

Table 411 – Unicast FDB entries

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Applica-tion	Bridge	ORG	Learn-ing	Meaning
Vendor	n	Receive [a]	Filter	No	Enabled	Interface MAC address
Vendor	n+1 – n+m	Receive [a]	Filter	No	Enabled	Port MAC addresses
Others	Others	—	—	—	—	

[a] If applicable

Table 412 – Multicast FDB entries

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Applica-tion	Bridge	ORG	Learn-ing	Meaning
01-0E-CF	00-00-00	Receive [a]	Forward	No	Enabled	With FrameID=0xFEFE used for DCP-Identify-ReqPDU
01-0E-CF	00-00-01	Receive [a]	Forward	No	Enabled	With FrameID=0xFEFC used for DCP-Hello-ReqPDU
01-0E-CF	00-00-20 – 00-00-3F	Receive [a]	Filter [g]	No	Enabled	With FrameID=0xFEFE used for filterable DCP-Identify-ReqPDU
01-0E-CF	00-01-01	Receive [a]	Filter [b]	No	Disabled	RT_CLASS_3 destination multicast address
01-0E-CF	00-01-02	Receive [a]	Filter [b]	No	Disabled	RT_CLASS_3 invalid frame multicast address
01-0E-CF	00-02-00 – 00-02-1F	Receive [a]	Forward [c]	No	Enabled	RT_CLASS_2 multicast communication address
01-0E-CF	00-02-20 – 00-02-FF	Receive [a]	Forward	No	Enabled	RT_CLASS_2 multicast communication address
01-0E-CF	00-04-00	Receive [a]	Forward [f]	No	Disabled	In conjunction with PTCP-AnnouncePDU and FrameID (=0xFF00) used for working clock synchronization
X3-XX-00	00-00-00	Receive [a]	Filter [b]	No	Disabled	Fast forwarding RT_CLASS_3 destination multicast address
01-80-C2	00-00-00	Receive [a]	Filter [d]	No [d]	Disabled	IEEE Std 802.1Q Rapid spanning tree protocol (RSTP)
01-00-5E	40-F8-00 – 40-FB-FF	Receive [a]	— [e]	No	Enabled	Used for multicast communication relations in conjunction with RT_CLASS_UDP
Others	Others	—	—	—	—	—

[a] If applicable.

[b] These frames shall be discarded by the MAC Relay if RT_CLASS_3 is supported, otherwise they should be discarded.

[c] Forwarding depends on the setting of MulticastBoundary.

[d] It is recommended to forward Rapid Spanning Tree Protocol (RSTP) frames in switches without RSTP support or switches with alternative media redundancy protocols like MRP.

[e] It is recommended to forward these frames unless IETF RFC 4604 (IGMPv3) is supported.

[f] Rule for implicit domain boundaries defined by PTCP applies.

[g] These frames shall be forwarded to ports connected to bridges (see 4.12.7). They shall only be forwarded to ports connected to end stations if the entry in the filtering database specifies the forwarding to this port.

Table 413 – Broadcast FDB entry

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Applica-tion	Bridge	ORG	Learn-ing	Meaning
FF-FF-FF	FF-FF-FF	Receive [a]	Forward	No	Enabled	Broadcast
Others	Others	—	—	—	—	—

[a] If applicable.

IEEE Std 802.1Q model aligns VID, FID and MST as shown in Table 414.

Table 414 – VID, FID and MSTID

VID	FID	MSTID	Traffic category	Meaning
Stream VLAN High	Stream High FDB	Traffic engineered TE-MST	Time-aware stream	—
Stream VLAN High Redundant	Stream High Redundant FDB	Traffic engineered TE-MST	Time-aware stream	—
Stream VLAN Low	Stream Low FDB	Traffic engineered TE-MST	Time-aware stream	—
Stream VLAN Low Redundant	Stream Low Redundant FDB	Traffic engineered TE-MST	Time-aware stream	—
Non-stream VLAN A	Non-stream FDB	Internal spanning tree IST/CIST	Stream, Traffic engineered non-stream, non-stream	Common and Internal Spanning Tree CIST together with RSTP support
Non-stream VLAN B	Non-stream FDB	Internal spanning tree IST/CIST	Non-stream	
Non-stream VLAN C	Non-stream FDB	Internal spanning tree IST/CIST	Non-stream	
Non-stream VLAN D	Non-stream FDB	Internal spanning tree IST/CIST	Non-stream	

4.12.4.7.3 Time-aware bridge

A time-aware system shall support VLANs.

Supported trees are:

– Non-stream VID
– Stream Low VID
– Stream Low Redundant VID
– Stream High VID
– Stream High Redundant VID
– IEC 62439-2
– IEEE Std 802.1AS Working Clock
– IEEE Std 802.1AS Working Clock Redundant
– IEEE Std 802.1AS Global Time
– IEEE Std 802.1AS Global Time Redundant
– IEEE Std 802.1AB

Table 415 shows whether the trees are with an FDB or assigned to LLC. Stream VIDs are used for category “Time-aware stream”, and non-stream VID A is used for category “Stream”.

Table 415 – Trees and FDBs

Associated spanning trees	Assigned FDB	Learning	Default Forwarding Rule	Default VID [decimal]
Non-stream VID A	FDB “Default”	Enabled, SVL [a]	Flooding	100
Stream VID High	FDB “High”	Disabled	Dropping	101
Stream VID High Redundant	FDB “High R”	Disabled	Dropping	102
Stream VID Low	FDB “Low”	Disabled	Dropping	103
Stream VID Low Redundant	FDB “Low R”	Disabled	Dropping	104
Non-stream VID B [b]	FDB “Default”	Enabled, SVL [a]	Flooding	105
Non-stream VID C [b]	FDB “Default”	Enabled, SVL [a]	Flooding	106
Non-stream VID D [b]	FDB “Default”	Enabled, SVL [a]	Flooding	107
IEC 62439-2 “MRP”	FDB “Default” and/or LLC	Disabled	—	—
IEEE Std 802.1AS Working Clock	LLC	Disabled	—	—
IEEE Std 802.1AS Working Clock “Hot standby redundancy”	LLC	Disabled	—	—
IEEE Std 802.1AS Global Time	LLC	Disabled	—	—
IEEE Std 802.1AS Global Time “Hot standby redundancy”	LLC	Disabled	—	—
IEEE Std 802.1AB	LLC	Disabled	—	—

[a] Shared with A, B, C and D

[b] Reserved for, e.g. passthrough traffic through a NME domain

For non-stream FDBs, Table 407 shows the expected number of possible entries, Table 408 the expected neighborhood for hash-based FDBs, and Table 409 the FDB attributes.

For stream FDBs, the expected number of DA-MAC address entries used together with four VLANs (High, High Red, Low and Low Red) are shown in Table 416, Table 417, and Table 418.

Table 416 – Number of stream FDB entries

# of VLANs	# of DA-MACs	Usage
4	2 048	Default Numbers of DA-MAC address entries used together with four VLANs (High, High Red, Low and Low Red) Same DA-MAC entries are used for all four VLANs (e.g. DA-MACs are rows and VLANs are columns of a table) Optimization: IEC/IEEE 60802-defined range of MAC addresses to allow “index model” for Table access.

Table 417 – Neighborhood for Stream entries

Neighborhood [a]	Usage
0	Default No neighborhood needed due to the selection of DestinationAddresses and the implementation of the FDB.
4	A neighborhood of four entries is used to store a learned entry if the hashed entry is already used.
8	A neighborhood of eight entries is used to store a learned entry if the hashed entry is already used.
[a] A neighborhood of x adjacent entries for the hashed index is checked when adding, seeking, or updating an entry in the FDB.	

Table 418 – FDB attributes for "Streams"

Flag	Value	Usage
Learning	No	Default Never learn a SA-MAC
Default Forwarding Rule	Discard	Default Discard if no explicit forwarding rule is stated
Shared VLAN Learning	No	Default Each VLAN learns individually

4.12.4.7.4 Non-time-aware bridge

A non-time-aware system may support VLANs.

Supported trees are:

– Default
– IEC 62439-2
– IEEE Std 802.1AS Global Time
– IEEE Std 802.1AS Global Time Redundant
– IEEE Std 802.1AB
– PTCP
– RT_CLASS_3

Table 419 shows whether the trees are with an FDB or assigned to LLC.

Table 419 – Trees and FDBs

Associated spanning trees	Assigned FDB	Learning	Default Forwarding Rule
Non-stream VID	FDB “Default”	Enabled, IVL	Flooding
IEC 62439-2	FDB “Default” and/or LLC	Disabled	Dropping
IEEE Std 802.1AS Global Time	LLC	Disabled	—
IEEE Std 802.1AS Global Time “Hot standby redundancy”	LLC	Disabled	—
IEEE Std 802.1AB	LLC	Disabled	—
PTCP	LLC	Disabled	—
RT_CLASS_3	FDB “RED_RELAY”	Disabled	Dropping

For non-stream FDBs, Table 407 shows the expected number of possible entries, Table 408 the expected neighborhood for hash based FDBs, and Table 409 the FDB attributes.

4.12.4.8 Active topology enforcement

See IEEE Std 802.1Q-2018, 8.6.1.

Active topology enforcement detects the assigned tree using Destination MAC, VID or protocol and enforces the rules of this tree.

Uses as input:

– Reception Port State
– Transmission Port State
– Filtering Database

4.12.4.9 Ingress

See IEEE Std 802.1Q-2018, 8.6.2.

Check if this receive port is part of an identified tree for this frame. If no tree is found, drop the frame.

4.12.4.10 Frame filtering

See IEEE Std 802.1Q-2018, 8.6.3.

Frame filtering applies the rules from the found tree and its assigned FDB to the frame to decide about the transmission port or ports.

4.12.4.11 Egress

See IEEE Std 802.1Q-2018, 8.6.4.

Check if this transmit port is part of an identified tree. Drop frame if VID is unknown.

4.12.4.12 Flow classification and metering

4.12.4.12.1 General

See IEEE Std 802.1Q-2018, 8.6.5, and Metro Ethernet Forum (MEF), Technical Specification 10.3.

The Ingress Rate Limiter or Flow Meter, for example, defined flow classification and metering, allows an ingress flow control to protect the resources of a bridge.

Ingress rate limiting is based on flow meters. The destination MAC address, TCI.VID, TCI.PCP and the selected MEF 10.3 models and parameters are used to define the behavior.

Metering for unicast, multicast and broadcast frames shall be supported to limit the load inside the domain due to traffic from outside the domain.

Network control protocols, for example protocols like IEEE Std 802.1AS or IEEE Std 802.1AB, are unaffected by these filters.

Table 420 shows the different groups of traffic handled by the flow meters, Table 421 shows the flow meter parameter used to handle the different groups of traffic, Table 422 the selection criterias, and Table 423 shows the flow meters.

Table 420 – Traffic grouping

Parameter	Envelope and exclusions	Comment
(A)	Best effort traffic	All traffic which is not identified as – Network control – Real-time – Time-aware stream
(B)	Real-time traffic	Real-time traffic
I	Excluded: Time-aware stream traffic	Time-aware stream traffic

Table 421 – Ingress rate limiter / Flow meter parameter

Parameter	Value	Comment
Committed information rate (CIR)	Meter dependent	Rate in bits per seconds
Committed burst size (CBS)	Meter dependent	Size in octets
Coupling flag (CF)	Meter dependent	Enables the coupling of flow meters
Color mode (CM)	Color-blind	Already existing coloring is ignored
Envelope	One or two envelopes (A) := Best effort envelope and (B) := RT_CLASS_X, RTA_CLASS_X envelope	Grouping of flow meters
Rank	CF0 to CF5	Sequence inside the grouping for bandwidth sharing
NOTE Excess Information Rate (EIR) and Excess Burst Size (EBS) are not used.		

Table 422 – Ingress rate limiter / Flow meter identifier

Parameter	Value
Unicast	Any unicast MAC address
Multicast	Selection of multicast MAC addresses, one, multiple or all Broadcast excluded
Broadcast	Broadcast MAC address
TCI.PCP	Selection TCs, one, multiple or all

Table 423 – Flow classification / Flow meter

Flow meter instance	Envelopes, filters and exclusions	Comment
Broadcast	(A) and (B) DA-MAC	Used to limit the bandwidth of broadcast based communication
Multicast	(A) and (B) DA-MAC I DA-MAC (all), TCI.VID (all), TCI.PCP (without RT_CLASS_STREAM)	Used to limit the bandwidth of multicast based communication
Unicast	(A) DA-MAC (all), TCI.VID (all), TCI.PCP (all) (B) DA-MAC (all), TCI.VID (all), TCI.PCP (without RT_CLASS_X and RTA_CLASS_X) I DA-MAC (all), TCI.VID (all), TCI.PCP (without RT_CLASS_STREAM)	Used to limit the bandwidth of unicast based communication
RT_CLASS_X	(B) DA-MAC (all), TCI.VID (all), TCI.PCP (only RT_CLASS_X)	Used to limit the bandwidth of unicast based RT_CLASS_1/_UDP communication
RTA_CLASS_X	(B) DA-MAC (all), TCI.VID (all), TCI.PCP (only RTA_CLASS_X)	Used to limit the bandwidth of unicast based RTA_CLASS_1/_UDP communication
Additional available, but unused instance One	Any DA-MAC, TCI.VID and TCI.PCP based filter	Reserved
Additional available, but unused instance Two	Any DA-MAC, TCI.VID and TCI.PCP based filter	Reserved
Additional available, but unused instance Three	Any DA-MAC, TCI.VID and TCI.PCP based filter	Reserved

4.12.4.12.2 Domain Boundary Port

Figure 124 and Table 424 show the used algorithms, filters, and IEEE Std 802.1Q-2018, 8.6.5 and MEF 10.3 parameters.

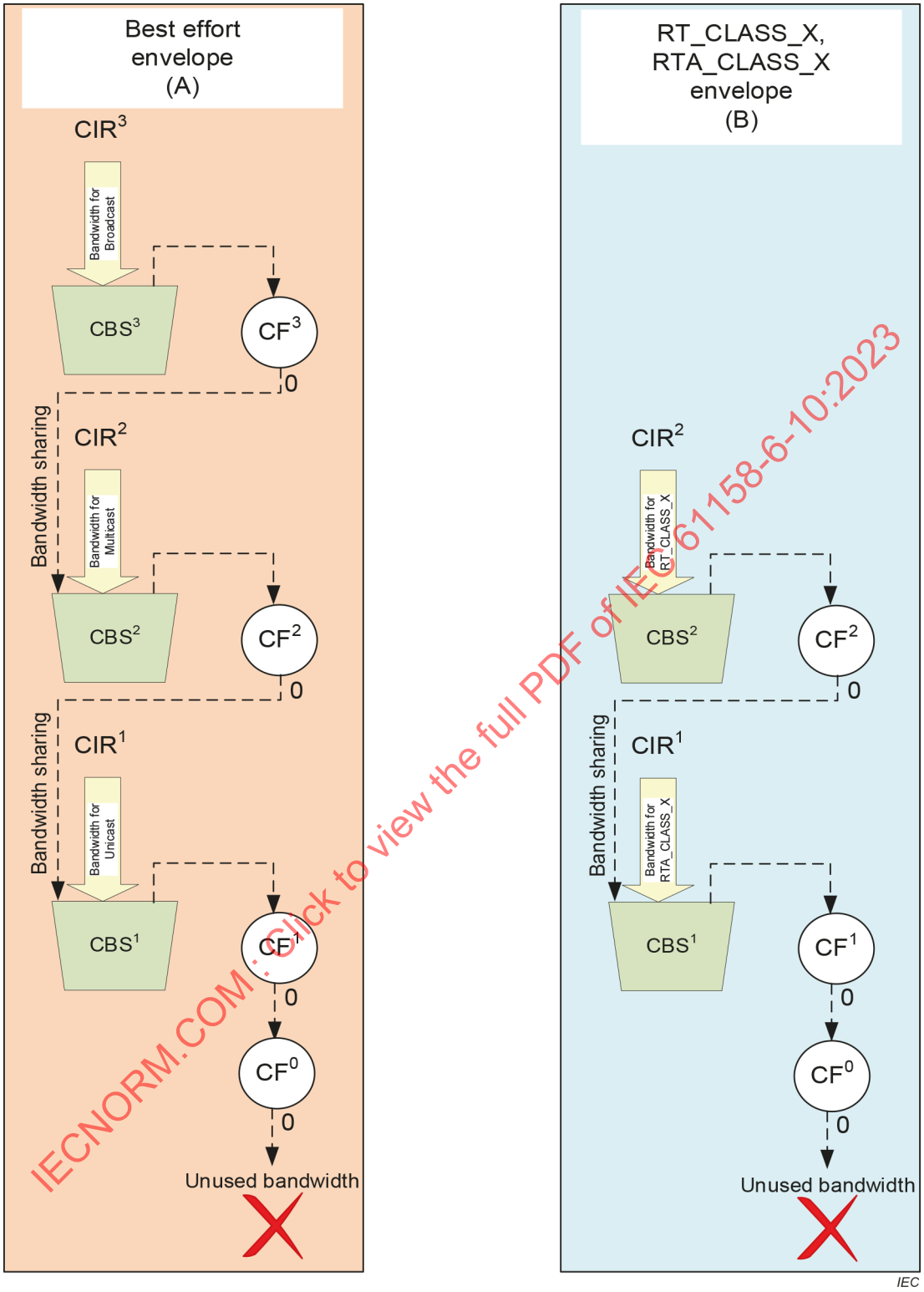


Figure 124 – Ingress rate limiter – Domain boundary

Table 424 – Flow classification and metering

Setup	Filter	Comment
(A) Best effort envelope	Included: All traffic	Only one envelope activated: 1.) Check DA-MAC address and TCI.PCP whether traffic is for (A) – if Broadcast, check available bandwidth, if exceeded, drop – if Multicast, check available bandwidth, if exceeded, drop – if Unicast, check available bandwidth, if exceeded, drop

(A)
Best effort envelope
and
(B)
RT_CLASS_X,
RTA_CLASS_X envelope
	Included:
	All traffic
		Two envelopes activated:
		1.) Check DA-MAC address and TCI.PCP whether traffic is for (A) or (B)
		if (A)
		– if Broadcast, check available bandwidth, if exceeded, drop
		– if Multicast, check available bandwidth, if exceeded, drop
		– if Unicast, check available bandwidth, if exceeded, drop
		If (B)
		– check available bandwidth, if exceeded, drop

(A)
Best effort envelope
and
(B)
RT_CLASS_X,
RTA_CLASS_X envelope
and
I
time-aware stream
	Included:
	All traffic
		Two envelopes activated and time-aware streams are used:
		1.) Check DA-MAC address and TCI.PCP whether traffic is for (A) or (B)
		if (A)
		– if Broadcast, check available bandwidth, if exceeded, drop
		– if Multicast, check available bandwidth, if exceeded, drop
		– if Unicast, check available bandwidth, if exceeded, drop
		If (B)
		– check available bandwidth, if exceeded, drop
		If I
		– forward

(A) Best effort envelope and I time-aware stream	Included: All traffic	One envelope activated and time-aware streams are used: 1.) Check DA-MAC address and TCI.PCP whether traffic is for (A) if (A) – if Broadcast, check available bandwidth, if exceeded, drop – if Multicast, check available bandwidth, if exceeded, drop – if Unicast, check available bandwidth, if exceeded, drop If I – forward

Table 425 and Table 426 show examples for the setup of ingress rate limiting as shown in Figure 124. It excludes any network control traffic, e.g. LLDP or gPTP.

Table 425 – Example values for flow classification and metering – (A) only

Flow meter	Filter	Comment
1	Unicast, TCI.PCP 0…7 Committed information rate: 10 Mbit/s Committed burst size: 4 000 octets Rank = 1 CF = coupling active	10 Mbit/s is equal to 1 250 000 octets/s
2	Multicast, TCI.PCP 0…7 Committed information rate: 5 Mbit/s Committed burst size: 2 000 octets Rank = 2 CF = coupling active	5 Mbit/s is equal to 625 000 octets/s
3	Broadcast, TCI.PCP 0…7 Committed information rate: 5 Mbit/s Committed burst size: 2 000 octets Rank = 3 CF = coupling active	5 Mbit/s is equal to 625 000 octets/s

Table 426 – Example values for flow classification and metering – (A) and (B)

Flow meter	Filter	Comment
(A) 1	Unicast, TCI.PCP 0…3, 5, 7 Committed information rate: 10 Mbit/s Committed burst size: 4 000 octets Rank = 1 CF = coupling active	10 Mbit/s is equal to 1 250 000 octets/s
(A) 2	Multicast, TCI.PCP 0…7 Committed information rate: 5 Mbit/s Committed burst size: 2 000 octets Rank = 2 CF = coupling active	5 Mbit/s is equal to 625 000 octets/s
(A) 3	Broadcast, TCI.PCP 0…7 Committed information rate: 5 Mbit/s Committed burst size: 2 000 octets Rank = 3 CF = coupling active	5 Mbit/s is equal to 625 000 octets/s
(B) 4	Unicast, TCI.PCP 6 Committed information rate: 10 Mbit/s Committed burst size: 4 000 octets Rank = 1 CF = coupling active	10 Mbit/s is equal to 1 250 000 octets/s

Flow meter	Filter	Comment
(B) 5	Unicast, TCI.PCP 4 Committed information rate: 10 Mbit/s Committed burst size: 4 000 octets Rank = 2 CF = coupling active	10 Mbit/s is equal to 1 250 000 octets/s

4.12.4.12.3 Link Speed Transition Port

Figure 125 and Table 427 show the used algorithms, filters, and IEEE Std 802.1Q-2018, 8.6.5 and MEF 10.3 parameters.

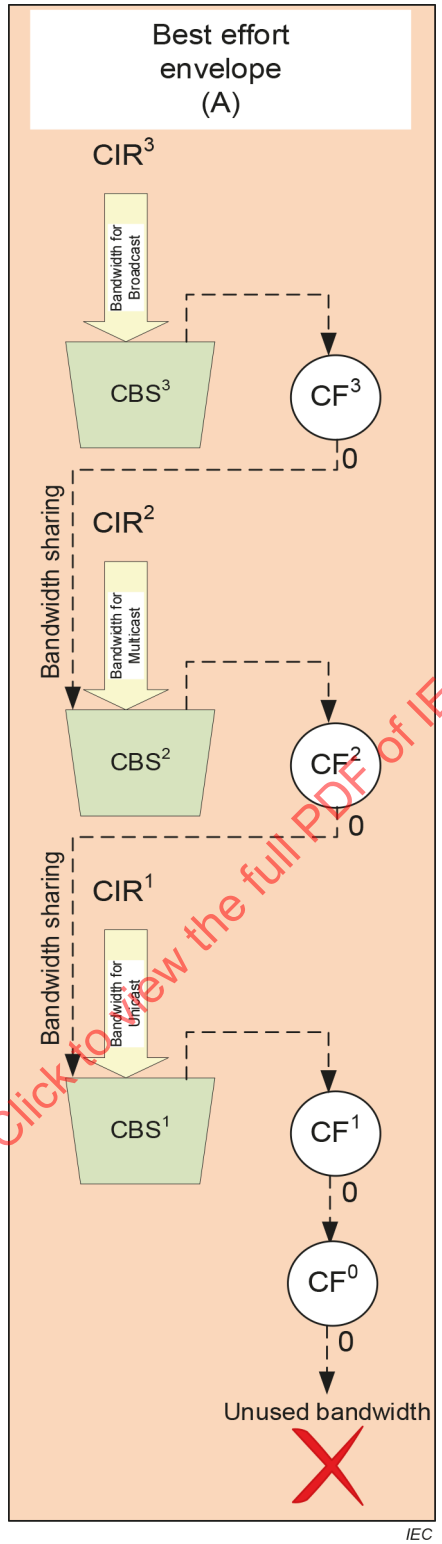


Figure 125 – Ingress rate limiter – Link speed transition

Table 427 – Flow classification and metering

Setup	Filter	Comment
(A) Best effort envelope	Included: All traffic	Only one envelope activated: 1.) Check DA-MAC address and TCI.PCP whether traffic is for (A) – if Broadcast, check available bandwidth, if exceeded, drop – if Multicast, check available bandwidth, if exceeded, drop – if Unicast, check available bandwidth, if exceeded, drop

(A)
Best effort envelope
and
(B)
RT_CLASS_X,
RTA_CLASS_X envelope
	Included:
	All traffic
		Two envelopes activated:
		1.) Check DA-MAC address and TCI.PCP whether traffic is for (A) or (B)
		if (A)
		– if Broadcast, check available bandwidth, if exceeded, drop
		– if Multicast, check available bandwidth, if exceeded, drop
		– if Unicast, check available bandwidth, if exceeded, drop
		If (B)
		– forward

(A)
Best effort envelope
and
(B)
RT_CLASS_X,
RTA_CLASS_X envelope
and
I
time-aware stream
	Included:
	All traffic
		Two envelopes activated and time-aware streams are used:
		1.) Check DA-MAC address and TCI.PCP whether traffic is for (A) or (B)
		if (A)
		– if Broadcast, check available bandwidth, if exceeded, drop
		– if Multicast, check available bandwidth, if exceeded, drop
		– if Unicast, check available bandwidth, if exceeded, drop
		If (B)
		– forward
		If I
		– forward

(A) Best effort envelope and I time-aware stream	Included: All traffic	One envelope activated and time-aware streams are used: 1.) Check DA-MAC address and TCI.PCP whether traffic is for (A) if (A) – if Broadcast, check available bandwidth, if exceeded, drop – if Multicast, check available bandwidth, if exceeded, drop – if Unicast, check available bandwidth, if exceeded, drop If I – forward

Table 428 shows examples for the setup of ingress rate limiting as shown in Figure 125. It excludes any network control traffic, e.g. LLDP or gPTP, unicast traffic with TCI.PCP values 4…7, and pruned multicasts defined by DCP.

Table 428 – Example values for flow classification and metering

Flow meter	Filter	Comment
1	Unicast, TCI.PCP 0…3 Committed information rate: 3 Mbit/s Committed burst size: 4 000 octets Rank = 1 CF = coupling active	3 Mbit/s is equal to 375 000 octets/s
2	Multicast, TCI.PCP 0…7 Committed information rate: 1 Mbit/s Committed burst size: 2 000 octets Rank = 2 CF = coupling active	1 Mbit/s is equal to 125 000 octets/s
3	Broadcast, TCI.PCP 0…7 Committed information rate: 1 Mbit/s Committed burst size: 2 000 octets Rank = 3 CF = coupling active	1 Mbit/s is equal to 125 000 octets/s

4.12.4.13 Queuing frames

4.12.4.13.1 General

See IEEE Std 802.1Q-2018, 8.6.6.

Identification of the traffic class and thus, of the queue in which the frame will be stored. Table 429 shows the available queues and as an example the assigned TCI values.

Each frame shall be assigned to a queue. This is done by TCI.PCP or in case of MAC service, as defined in IEEE Std 802.1AC, directly.

Table 429 – Queues and TCI

Queue	TCI.PCP / untagged frames	Comment
0	0, Untagged frames	Used for tagged or untagged frames
1	1, Untagged frames	Used for tagged or untagged frames
2	2, Untagged frames	Used for tagged or untagged frames
3	3, Untagged frames	Used for tagged or untagged frames
4	4, Untagged frames	Used for tagged or untagged frames
5	5, Untagged frames	Used for tagged or untagged frames
6	6, Untagged frames	Used for tagged or untagged frames
7	7, Untagged frames	Used for tagged or untagged frames

4.12.4.13.2 Model

Each Queue Frame entity of a bridge implements a resource management following the principles shown in Figure 126.

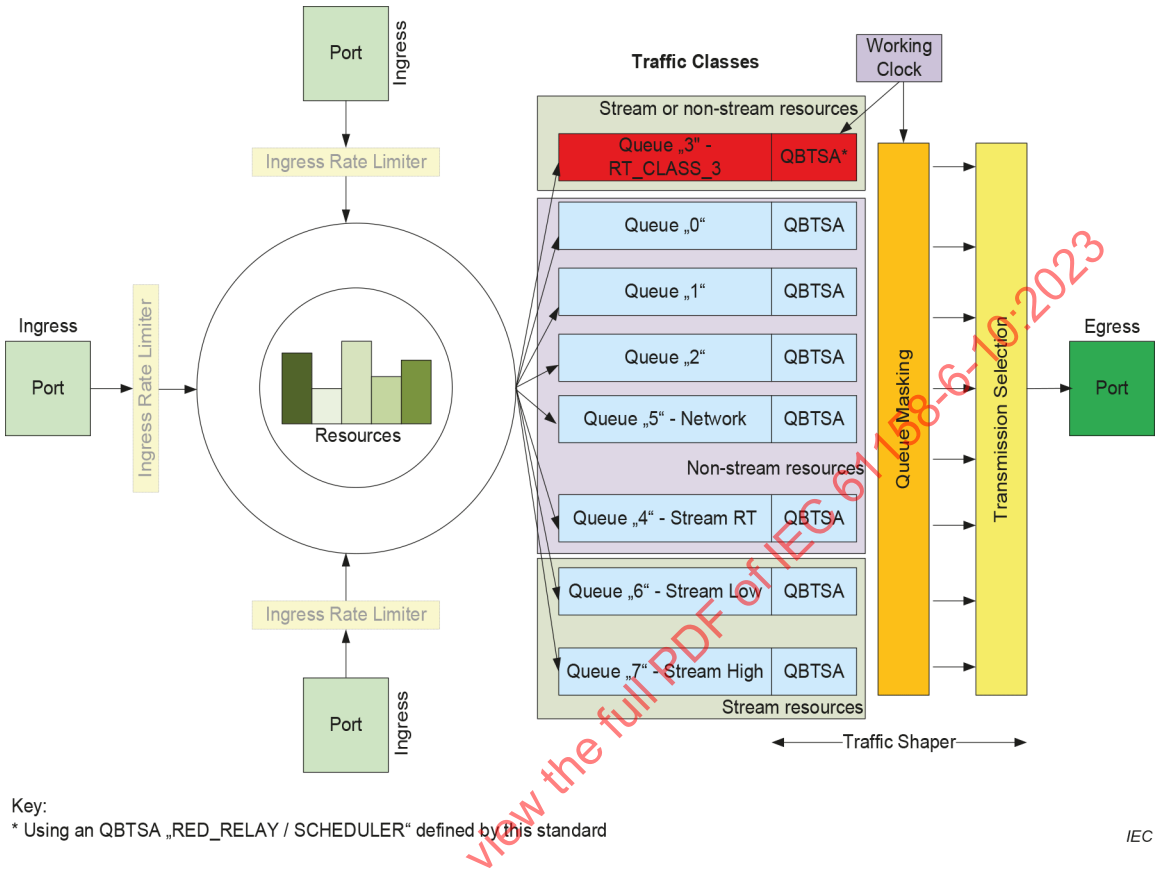


Figure 126 – Schematic traffic flow model of a bridge

4.12.4.13.3 Resources

4.12.4.13.3.1 General

The bridge shall provide and organize its resources in a way that ensures robustness for the traffic defined in this document.

Resources are needed to cover the following four topics for

- Time-aware systems, and
  - Zero congestion loss for time-aware streams through interference
  - Zero congestion loss for non-stream classes through interference with stream classes
  - Congestion loss prevention for non-time-aware streams due to non-stream traffic
  - Congestion loss prevention for internal non-stream traffic due to external non-stream traffic
- Non-time-aware systems
  - Zero congestion loss for RT_CLASS_3 through interference
  - Zero congestion loss for other traffic through interference with RT_CLASS_3
  - Congestion loss prevention for RT_CLASS_1/_2/_UDP due to non-real-time traffic
  - Congestion loss prevention for internal non-real-time traffic due to external traffic

The frame resources for the queues are allocated from a bridge internal “resource pool”. This “resource pool”, if possible managed with port granularity, contains dedicated frame resources for each queue and global frame resources available for all queues.

The amount of available frame resources shall cover the times in which the transmission of frames from some queues is blocked, without frame dropping due to the used traffic shaping.

4.12.4.13.3.2 Bridges

For bridge memory calculation Formula (49) applies. It assumes that all ports use the same link speed and sums up the needed frame resource to avoid congestion loss for non-stream traffic classes through interference with stream traffic classes for a bridge implementation.

Memory sizes for congestion loss prevention for non-real-time traffic classes due to interference with real-time traffic classes are calculated similarly.

Bridges with active link speed transition ports require a more complex calculation. In this case the summation of the different port/blocking time/link speed combinations is needed.

$$\text{MinimumFrameMemory} = (\text{NumberOfPorts} - 1) \times \text{MaxPortBlockingTime} \times 1 / \text{Duration[Data rate]} \quad (49)$$

where

MinimumFrameMemory	is the minimum amount (number of octets) of frame buffer needed to avoid frame loss from non-stream traffic due to streams blocking egress ports.
NumberOfPorts	is the number of ports of the bridge without the management port.
MaxPortBlockingTime	is the intended maximum blocking time (e.g. 200 µs for a 1 ms interval) of ports due to number of streams and/or gating windows size.
Duration[Data rate]	is the duration of an octet for the intended link speed (e.g. 8 ns for 1 Gbit/s) of the ports.

Table 430, Table 431, Table 432, Table 433, Table 434 and

Table 435 show the resulting values from Formula (49) for different link speeds.

The traffic from the management port to the network needs an adequate share of the bridge resources or its own pool to ensure the required injection performance into the network. This memory is not covered by this calculation.

Bridge based, port based, or queue-based frame resource management can solve the resource protection requirements.

A per port frame resource management leads to the same amount of memory but reduces the flexibility to use free frame resources for other ports.

A per queue per port frame resource management would increase the needed amount of frame resources dramatically (multiplied by the number of queues to be covered), almost without any benefit.

EXAMPLE 1 all queues shared (per port) frame resource:
100 Mbit/s, 2 Ports, and 6 queues

Needed memory := 6,25 Koctets * 2 := 12,5 Koctets

EXAMPLE 2 per queue frame resource:
100 Mbit/s, 2 Ports, and 6 queues

Needed memory := 6,25 Koctets * 6 * 2:= 75 Koctets

It is not possible to define which queue is needed during the “stream port blocking” period.

Table 430 – MinimumFrameMemory for 10 Mbit/s (50 % @ 8 ms)

# of ports	MinimumFrameMemory [Kbytes]	Comment
1	0	The memory at the management port is not covered by Formula (49)
2	5	All frames received during the 50 % @ 8 ms := 4 ms at one port needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
3	10	All frames received during the 50 % @ 8 ms := 4 ms at two ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
4	15	All frames received during the 50 % @ 8 ms := 4 ms at three ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
Other	—	Shall be calculated according to the Formula (49).

Table 431 – MinimumFrameMemory for 100 Mbit/s (50 % @ 1 ms)

# of ports	MinimumFrameMemory [Kbytes]	Comment
1	0	The memory at the management port is not covered by Formula (49)
2	6,25	All frames received during the 50 % @ 1 ms := 500 µs at one port needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
3	12,5	All frames received during the 50 % @ 1 ms := 500 µs at two ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
4	18,75	All frames received during the 50 % @ 1 ms := 500 µs at three ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
Other	—	Shall be calculated according to the Formula (49).

Table 432 – MinimumFrameMemory for 1 Gbit/s (20 % @ 1 ms)

# of ports	MinimumFrameMemory [Kbytes]	Comment
1	0	The memory at the management port is not covered by Formula (49)
2	25	All frames received during the 20 % @ 1 ms := 200 µs at one port needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
3	50	All frames received during the 20 % @ 1 ms := 200 µs at two ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
4	75	All frames received during the 20 % @ 1 ms := 200 µs at three ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
Other	—	Shall be calculated according to the Formula (49).

Table 433 – MinimumFrameMemory for 2,5 Gbit/s (10 % @ 1 ms)

# of ports	MinimumFrameMemory [Kbytes]	Comment
1	0	The memory at the management port is not covered by Formula (49)
2	31,25	All frames received during the 10 % @ 1 ms := 100 µs at one port needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
3	62,5	All frames received during the 10 % @ 1 ms := 100 µs at two ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
4	93,75	All frames received during the 10 % @ 1 ms := 100 µs at three ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
Other	—	Shall be calculated according to the Formula (49).

Table 434 – MinimumFrameMemory for 5 Gbit/s (5 % @ 1 ms)

# of ports	MinimumFrameMemory [Kbytes]	Comment
—	—	Intentionally left blank

Table 435 – MinimumFrameMemory for 10 Gbit/s (5 % @ 1 ms)

# of ports	MinimumFrameMemory [Kbytes]	Comment
1	0	The memory at the management port is not covered by Formula (49)
2	62,5	All frames received during the 5 % @ 1 ms := 50 µs at one port needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
3	125	All frames received during the 5 % @ 1 ms := 50 µs at two ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
4	187,5	All frames received during the 5 % @ 1 ms := 50 µs at three ports needed to be forwarded to the other port are stored during the allocation of this port due to stream transmission.
Other	—	Shall be calculated according to the Formula (49).

4.12.4.13.3.3 Egress port resource management model

Devices implementing both time-aware system mode and non-time-aware system mode need to support the maximum number of resources according to Table 436 and Table 437.

Figure 127 shows the values from Table 436 for one port of a bridge in time-aware system setup.

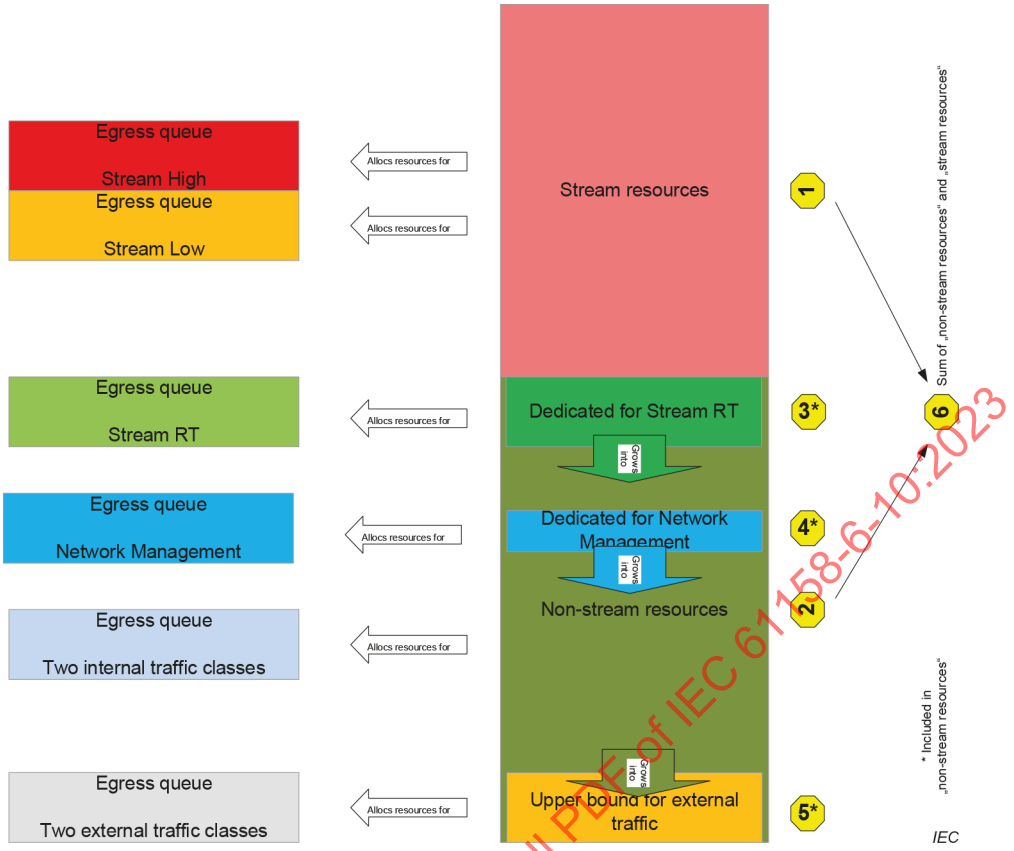


Figure 127 – Time-aware system – Egress port resource model of a bridge

Table 436 – Minimum Frame Buffer Memory for one egress port (time-aware system)

Portion	10 Mbit/s [ms]	100 Mbit/s [µs]	1 Gbit/s [µs]	2,5 Gbit/s [µs]	10 Gbit/s [µs]
1	4	500	200	100	50
2	6,5	750	250	125	75
3	2,5	250	50	25	5
4	1,25	125	12,5	5	1,25
5	2,5	375	50	25	5
6	10,5 (13 125 Octets)	1 250 (15 625 Octets)	450 (56 250 Octets)	225 (70 312,5 Octets)	125 (156 250 Octets)

Figure 128 shows the values from Table 437 for one port of a bridge in non-time-aware system setup.

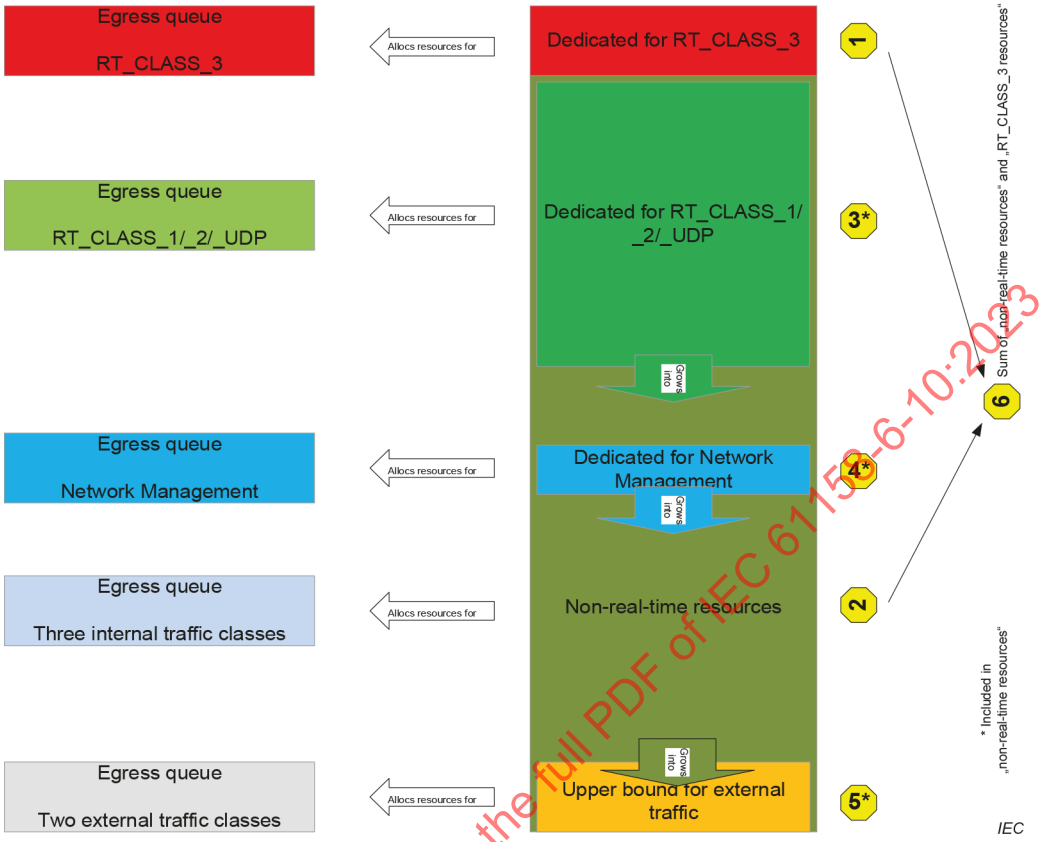


Figure 128 – Non-time-aware system − Egress port resource model of a bridge

Table 437 – Minimum Frame Buffer Memory for one egress port (Non-time-aware system)

Portion	10 Mbit/s [ms]	100 Mbit/s [µs]	1 Gbit/s [µs]	2,5 Gbit/s [µs]	10 Gbit/s [µs]
1	—	250	—	—	—
2	6,5	750	250	125	55
3	2,5	250	50	25	5
4	1,25	125	12,5	5	1,25
5	2,5	375	50	25	5
6	6,5 (8 125 Octets)	1 000 (12 500 Octets)	250 (31 250 Octets)	125 (39 062,5 Octets)	55 (68 750 Octets)

4.12.4.13.3.4 Possible optimizations for two port bridged end stations

Figure 140 shows a bridged end station which offers possibilities to optimize the minimum frame buffer resources for external ports. These optimizations consider that the end station portion of a bridged end station controls the interference it creates to these ports.

Thus, some of the values shown in Figure 127 and Table 436, and Figure 128 and Table 437 may be reduced depending on the end station requirements.

EXAMPLE 1 End station with four streams within sum 4,5 kByte of data at 1 Gbit/s:

Stream (High and Low) interference 4,5 kByte + 1,5 kByte (one maximum frame)

Memory needed to buffer 50 µs to avoid congestion loss

This optimization is only possible, if the end station does not already produce the maximum allowed stream traffic itself.

4.12.4.13.4 Time-aware bridge

4.12.4.13.4.1 General

Identify the traffic class and thus, the queue in which the frame will be stored. Table 438 shows the model selection for the different link speeds.

IEEE Std 802.1AC defines a local interface (e.g. a management port) which allows to inject frame into any queue. This is mainly used for network control.

Untagged, but no network control protocol, frames are handled like frames containing a PCP value of zero.

Table 438 – Model selection

Link speed	Traffic Class requirement	Selection	Comment
10 Mbit/s	HIGH and LOW and/or RT	Queue Masking required	Queue masking for HIGH to support the required latency values. HIGH, LOW and RT are express and use Cut through
10 Mbit/s	LOW and/or RT	Without Queue masking	LOW and RT are express and use Cut through
100 Mbit/s	HIGH and LOW and/or RT	Queue Masking required	Queue masking for HIGH to support the required latency values. HIGH, LOW and RT are express and use Cut through
100 Mbit/s	LOW and/or RT	Without Queue masking	LOW and RT are express and use Cut through
1 Gbit/s	HIGH and LOW and/or RT	Without Queue masking	HIGH is express and use Cut through and LOW and RT are preemptable and use Store&Forward
1 Gbit/s	LOW and/or RT	Without Queue masking	LOW and RT are express and use Cut through
> 1 Gbit/s	HIGH and LOW and/or RT or LOW and/or RT	Without Queue masking	No Preemption and no Cut through is expected
NOTE IEEE Std 802.1Q and IEEE Std 802.3 working on an amendment to support at least 10BASE-T1L for preemption.			

4.12.4.13.4.2 Without Queue Masking

Identification of the traffic class and thus, of the queue in which the frame will be stored. Table 439 shows the queue usage.

Table 439 – Queue usage – time-aware bridge – without queue masking

Queue	TCI.PCP	Stream	Preemption	Mask [Group]	QBTSA	Cut through	Comment
0	0	—	Preemptable	—	Strict priority	—	Non-stream: BEL Inter region traffic, passing by. Other protocols
1	1	—	Preemptable	—	Strict priority	—	Non-stream: BEH Inter region traffic, passing by. Other protocols
2	2	—	Preemptable	—	Strict priority	—	Traffic engineered, non-stream: CO Protocols used for connection establishment, records and control Domain internal traffic
3	3	—	Preemptable	—	Strict priority	—	Traffic engineered, non-stream: EV Acyclic transmitted frames rated by this document as RTA_CLASS_X / Alarm
4	4	RT	Express / Preemptable	—	Strict priority	—	Stream, class RT Cyclic transmitted frames rated by this document as “Stream, class RT” for nodes inside or outside of the time-aware domain

Queue	TCI.PCP	Stream	Preemption	Mask [Group]	QBTSA	Cut through	Comment
5	7	—	Preemptable	—	Strict priority	—	Traffic engineered, non-stream: NW Network control e.g. synchronization, neighborhood
6	5	LOW	Express / Preemptable	—	Strict priority	Yes / No	Time-aware stream, class LOW Cyclic transmitted frames rated by this document as “Time-aware stream, class LOW”
7	6	HIGH	Express	—	Strict priority	Yes	Time-aware stream, class HIGH Cyclic transmitted frames rated by this document as “Time-aware stream, class HIGH”

4.12.4.13.4.3 With Queue Masking

Identification of the traffic class and thus, of the queue in which the frame will be stored. Table 440 shows the queue usage and Figure 131 the queue mask usage.

Table 440 – Queue usage – time-aware bridge – with queue masking

Queue	TCI.PCP	Stream	Preemption	Mask [Group]	QBTSA	Cut through	Comment
0	0	—	Preemptable	Other	Strict priority	—	Non-stream: BEL Inter region traffic, passing by. Other protocols
1	1	—	Preemptable	Other	Strict priority	—	Non-stream: BEH Inter region traffic, passing by. Other protocols
2	2	—	Preemptable	Other	Strict priority	—	Traffic engineered, non-stream: CO Protocols used for connection establishment, records and control Domain internal traffic
3	3	—	Preemptable	Other	Strict priority	—	Traffic engineered, non-stream: EV Acyclic transmitted frames rated by this document as RTA_CLASS_X / Alarm
4	4	RT	Preemptable	Other	Strict priority	—	Stream: RT Cyclic transmitted frames rated by this document as “Stream, class RT” for nodes inside or outside of the time-aware domain
5	7	—	Preemptable	Other	Strict priority	—	Traffic engineered, non-stream: NW Network control e.g. synchronization, neighborhood
6	5	LOW	Express	Low	Strict priority	Yes	Time-aware stream: LOW Cyclic transmitted frames rated by this document as “Time-aware stream, class LOW”
7	6	HIGH	Express	High	Strict priority	Yes	Time-aware stream: HIGH Cyclic transmitted frames rated by this document as “Time-aware stream, class HIGH”

4.12.4.13.5 Non-time-aware bridge

4.12.4.13.5.1 General

Identify the traffic class and thus, the queue in which the frame will be stored.

IEEE Std 802.1AC defines a local interface (e.g. a management port) which allows to inject frame into any queue. This is mainly used for network control.

Untagged, but no network control protocol, frames are handled like frames containing a PCP value of zero.

4.12.4.13.5.2 Without RT_CLASS_3

Identification of the traffic class and thus, of the queue in which the frame will be stored. Table 441 shows the queue usage.

Table 441 – Queue usage – non-time-aware bridge – without RT_CLASS_3

Queue	TCI.PCP	RED_-RELAY	Pre-emption	Mask [Group]	QBTSA and IEC 61158-x-10 Fragmentation	Cut through	Comment
0	0	—	—	—	Strict priority	Yes	Best effort low
1	1	—	—	—	Strict priority	Yes	Best effort high
2	2	—	—	—	Strict priority	Yes	Protocols used for connection establishment, records and control Domain internal traffic
3	3	—	—	—	Strict priority	Yes	—
4	4	—	—	—	Strict priority	Yes	—
5	5	—	—	—	Strict priority	Yes	Acyclic transmitted frames rated by this document as RTA_CLASS_X / Alarm
6	6	—	—	—	Strict priority	Yes	Cyclic transmitted frames rated by this document as RT_CLASS_1, _2, _UDP
7	7	—	—	—	Strict priority	Yes	Network control e.g. synchronization, neighborhood

4.12.4.13.5.3 With RT_CLASS_3

Identification of the traffic class and thus, of the queue in which the frame will be stored. Table 442 shows the queue usage and Figure 132 the RED_RELAY usage.

Table 442 – Queue usage – non-time-aware bridge – with RT_CLASS_3

Queue	TCI.PCP	RED_-RELAY	Pre-emption	Mask [Group]	QBTSA and IEC 61158-x-10 Fragmentation	Cut through	Comment
0	0	—	—	Other	Strict priority w/o fragmentation	Yes	Best effort low
1	1	—	—	Other	Strict priority w/o fragmentation	Yes	Best effort high
2	2	—	—	Other	Strict priority w/o fragmentation	Yes	Protocols used for connection establishment, records and control Domain internal traffic
3	3 and 4	—	—	Other	Strict priority w/o fragmentation	Yes	—
4	5	—	—	Other	Strict priority w/o fragmentation	Yes	Acyclic transmitted frames rated by this document as RTA_CLASS_X / Alarm
5	6	—	—	Other	Strict priority	Yes	Cyclic transmitted frames rated by this document as RT_CLASS_1, _2, _UDP
6	7	—	—	Other	Strict priority	Yes	Network control e.g. synchronization, neighborhood
7	—	Yes	—	RED [a]	RT_CLASS_3 scheduler	Yes	Cyclic transmitted frames rated by this document as RT_CLASS_3

[a] When active masks all other queues.

4.12.4.14 Queue management

See IEEE Std 802.1Q-2018, 8.6.7.

This module controls the content of the queue and deletes frames if:

– they are forwarded to the MAC or
– they are aged or
– the port for a given frame is no longer part of the active topology or
– necessary to ensure overall quality of service.

4.12.4.15 Queue based transmission selection

4.12.4.15.1 General

See IEEE Std 802.1Q-2018, 8.6.7.

This module provides per queue a “frame available” flag which is used by “Preemptable transmission selection” and “Normal/Express transmission selection”.

This flag is created by the QBTSA per queue using

– “strict priority” QBTSA, which signals “frame available” if queue contains a frame
– “other, e.g. CBSA” QBTSA, which signals “frame available” if a frame could be transmitted according to the selected QBTSA

4.12.4.15.2 Time-aware bridge

Table 439 and Table 440 show the assigned queue based transmission selection.

4.12.4.15.3 Non-time-aware bridge

4.12.4.15.3.1 Without RT_CLASS_3

Table 441 shows the assigned queue-based transmission selection.

4.12.4.15.3.2 With RT_CLASS_3

Table 442 shows the assigned queue-based transmission selection.

The QBTSA “Fragmentation” and “RT_CLASS_3 scheduler” are defined in this document. They use WorkingClock and Phase Control to segment frames and affect the Queue Masking outside of the RED PHASE.

4.12.4.16 Queue masking for bridges

4.12.4.16.1 General

See IEEE Std 802.1Q.

The “Queue based transmission selection” provides per queue a “frame available” flag which may be overwritten by Time Aware Shaper and the eMAC (used together with preemption).

Queue assignment for bridges and end stations can differ. End stations transmit time-aware streams and streams as a burst and subsequently the non-stream traffic. Bridges prioritize network control over streams during forwarding.

The WorkingClock is used as time information if time is used as masking criteria (Gate control) as shown in Figure 129.

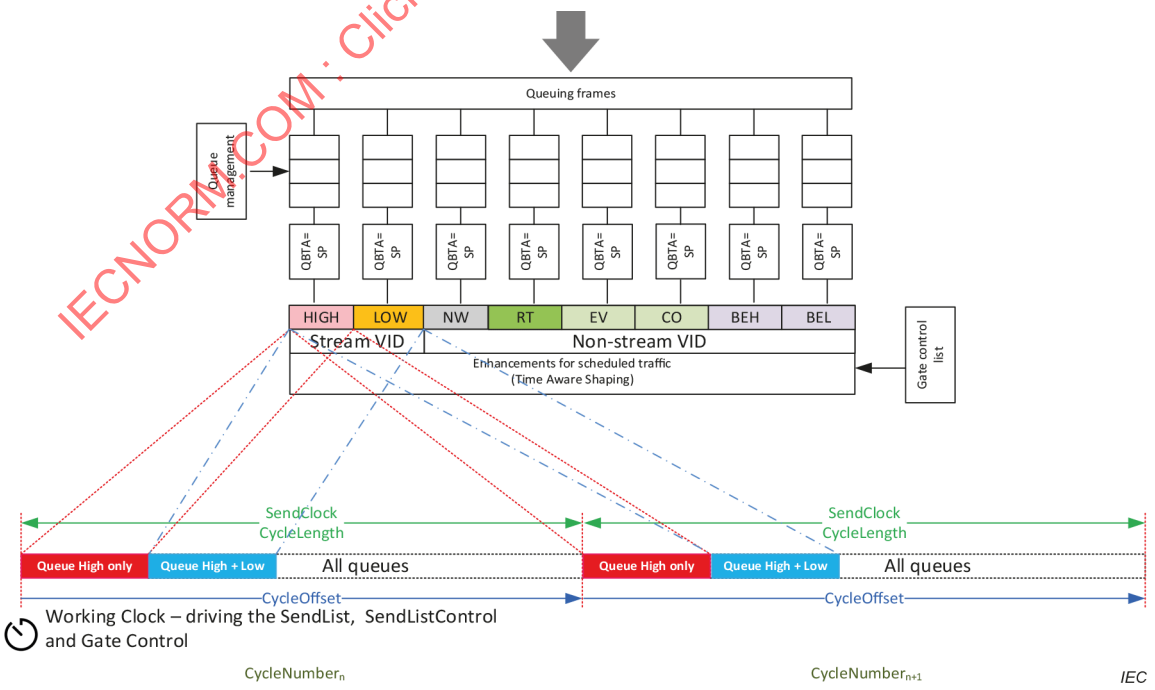


Figure 129 – Bridge queue masking usage model

The WorkingClock is used as time information if time is used as masking criteria (RED_RELAY) as shown in Figure 130.

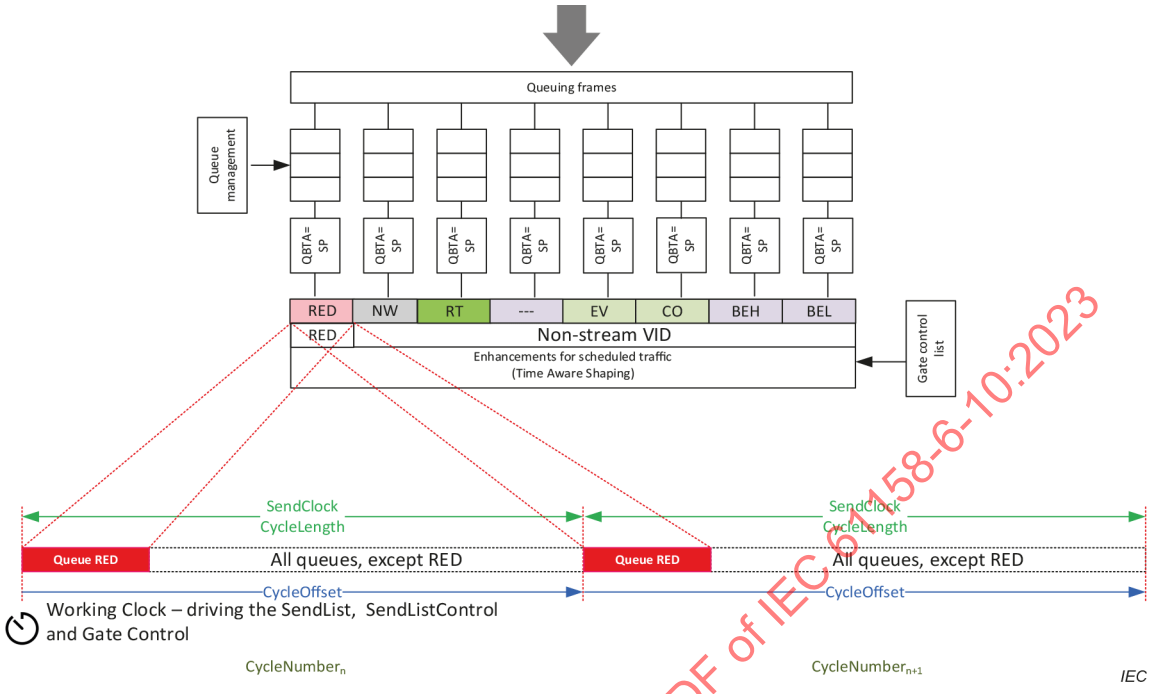


Figure 130 – RED_RELAY – Bridge queue masking usage model

Network access at end stations or bridged end stations shall be time-aware and access the network synchronized to the WorkingClock in the order shown in Figure 129 and Figure 130.

4.12.4.16.2 Time-aware bridge

4.12.4.16.2.1 Link speed 10 Mbit/s

Used as specified by IEEE Std 802.1Q.

If queue masking is used, then 4.12.4.16.2.2 applies.

4.12.4.16.2.2 Link speed 100 Mbit/s

Used as specified by IEEE Std 802.1Q.

The WorkingClock together with the Gate Control is used as queue masking criteria for the accessible queues.

EXAMPLE: With Gate Open for one queue only this queue is visible for the transmission selection.

Figure 129 and Figure 131 show the used TAS setup. High queue is never masked, Low queue only during “High queue only” time and all other queues during “High and low queue only” time.

Hold/release to ensure in-time network access of the “High queue only” traffic is only needed to protect the begin of the gating cycle / “High queue only” time if preemption is used for other queues and residual time of preemptable frames is not handled automatically by the hardware.

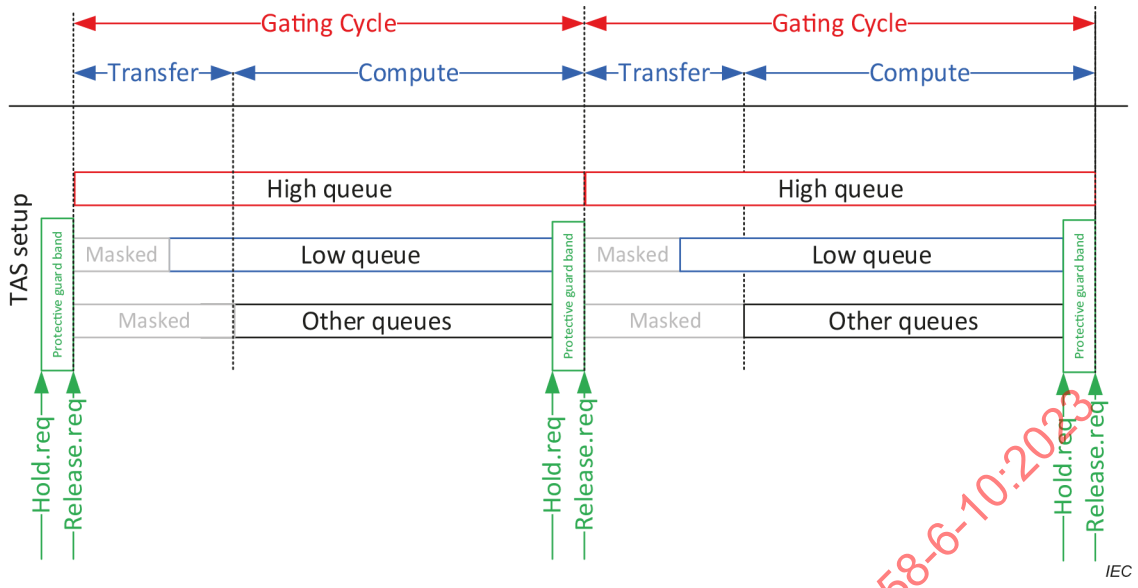


Figure 131 – TAS setup – Bridge queue masking model

4.12.4.16.2.3 Link speed ≥ 1 Gbit/s

Used as specified by IEEE Std 802.1Q.

The use of queue masking is not intended.

4.12.4.16.3 Non-time-aware bridge

4.12.4.16.3.1 Without RT_CLASS_3

Used as specified by IEEE Std 802.1Q.

4.12.4.16.3.2 With RT_CLASS_3

The queue for RT_CLASS_3 is handled comparable to a stream queue.

The WorkingClock together with the Phase Control is used as masking criteria for the accessible queues.

With the start of the RED PHASE only the RED/RT_CLASS_3 queue is visible for the transmission selection as shown in Figure 132.

As soon as the RED PHASE ends, the RED/RT_CLASS_3 queue is invisible for the transmission selection.

Outside of the RED PHASE, used as specified by IEEE Std 802.1Q

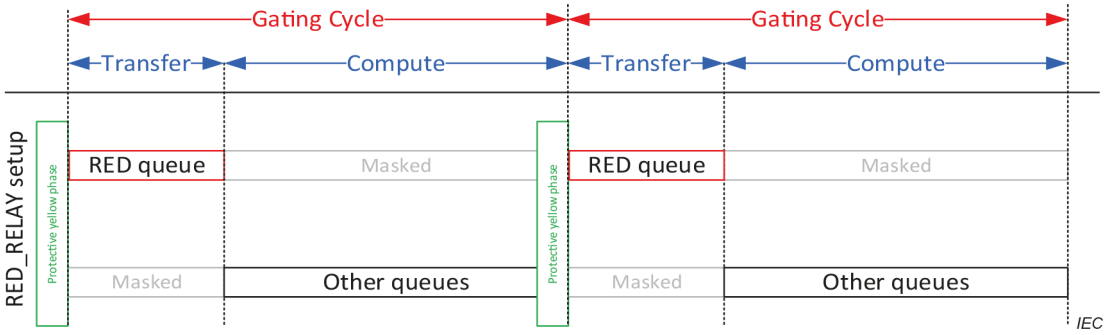


Figure 132 – RED_RELAY setup – Queue masking model

4.12.4.17 Transmission selection

4.12.4.17.1 Normal transmission selection

See IEEE Std 802.1Q-2018, 8.6.8.

This module fetches a frame from the queue with the highest priority traffic class, if “frame available” flag equals true. This process is repeated after each frame.

Transmission selection always seeks the next frame by starting with the queue with the highest number assigned to Normal.

4.12.4.17.2 Express transmission selection

See IEEE Std 802.1Q-2018, 8.6.8.

This module fetches a frame from the express queue if “frame available” flag equals true. This process is repeated after each frame.

Transmission selection always seeks the next frame by starting with the queue with the highest number assigned to Express.

4.12.4.17.3 Preemption

See IEEE Std 802.1Q-2018, 6.7.2 and IEEE Std 802.1Q-2018, 8.6.8.

Table 443 shows the required frame preemption parameters.

Table 443 – Preemption parameter

MinimumInitialFragmentSize	Comment
64 octets	Mandatory The transmit port is blocked for 64 + 64 octets
128 octets	The transmit port is blocked for 128 + 64 octets
192 octets	The transmit port is blocked for 192 + 64 octets
256 octets	The transmit port is blocked for 256 + 64 octets

4.12.4.18 MAC

4.12.4.18.1 General

See IEEE Std 802.1Q and IEEE Std 802.3.

4.12.4.18.2 Express or Normal MAC

Transmission or reception of express frames.

4.12.4.18.3 Preemptible MAC

Transmission or reception of preemptable frames.

4.12.4.18.4 MAC Control

MAC Control frames like PAUSE do interfere with the latency and jitter requirements. It shall be possible to disable the support of this protocol.

Vendor specific flow control between management port and bridge may be supported to reduce the needed frame buffer memory of the bridge.

4.12.4.18.5 Energy Efficient Ethernet

EEE do interfere with the latency and jitter requirements. It shall be possible to disable the support.

4.12.4.18.6 Packet size

Packet size according to IEEE Std 802.3-2018, 3.2.7, including envelope frame, shall be supported.

It shall be possible to disable [Discard on receive / invalid packet] the support of frames bigger than defined in IEEE Std 802.3.

4.12.4.18.7 MAC delay

The expected MAC delays are measured from the first bit of the first byte at the SHIM to the first bit of the first byte behind the MAC as shown in Figure 44.

4.12.4.19 PHY

4.12.4.19.1 General

See IEEE Std 802.3.

Usable Media Types need to support

– IEEE Std 802.1AS,
– IEEE Std 802.1AB, and
– full-duplex transmission.

4.12.4.19.2 Media Types

Table 444 shows the required MAU types. In addition, the PHYs shall fulfill the requirements according to the definition in Figure 44.

Table 444 – Media Types

MAU type Media	MAU type Link speed	Non-time-aware system Requirement	Time-aware system Requirement
Copper	10 Mbit/s	Optional	Optional
	100 Mbit/s	Mandatory [a]	Optional
	1 Gbit/s	Optional	Optional
	2,5 Gbit/s	Optional	Optional
	Other	Optional	Optional
Fiber	100 Mbit/s	Mandatory [a]	Optional
	1 Gbit/s	Optional	Optional
	2,5 Gbit/s	Optional	Optional
	Other	Optional	Optional
POF	100 Mbit/s	Mandatory [a]	Optional
	1 Gbit/s	Optional	Optional
	Other	Optional	Optional
Others	Others	Optional	Optional

[a] At least one of these MAUTypes, Optional if Single Pair Ethernet (SPE) for 10 Mbit/s is supported.

4.12.4.20 Bridge Delay

The forwarding or bridging delay in both time-aware and non-time-aware systems shall be kept near the theoretical minimum according to the definition of Figure 44.

4.12.4.21 Examples

Figure 133, Figure 134, Figure 135, Figure 136, Figure 137, and Figure 138 show the application model of a bridge (for simplicity with only two external ports) with an internal or management port hosting an end station according to the IEEE Std 802.1Q.

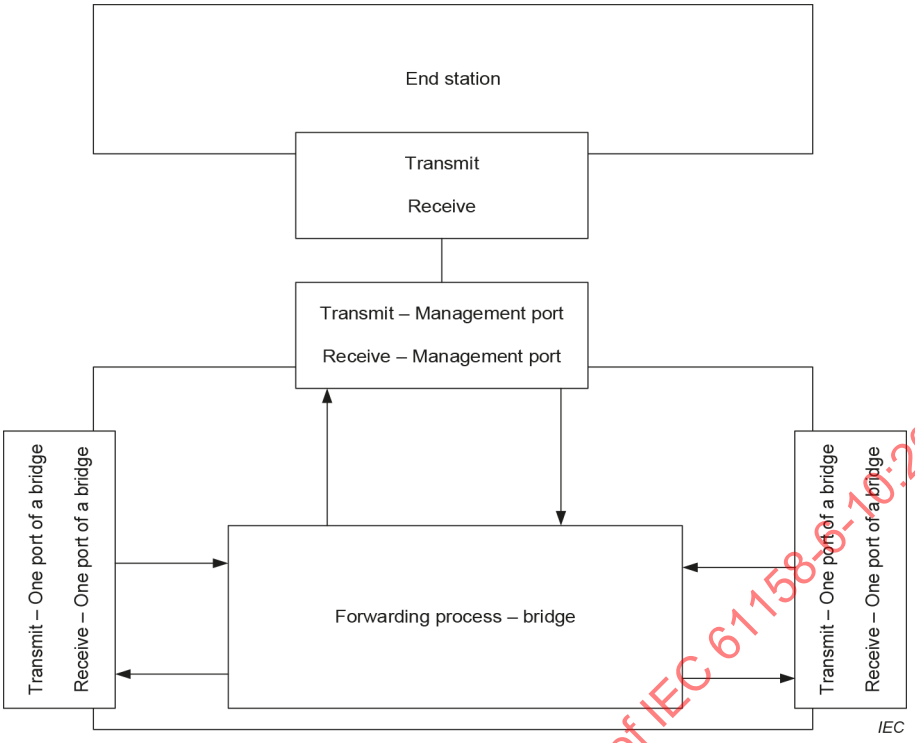


Figure 133 – Bridge with end station

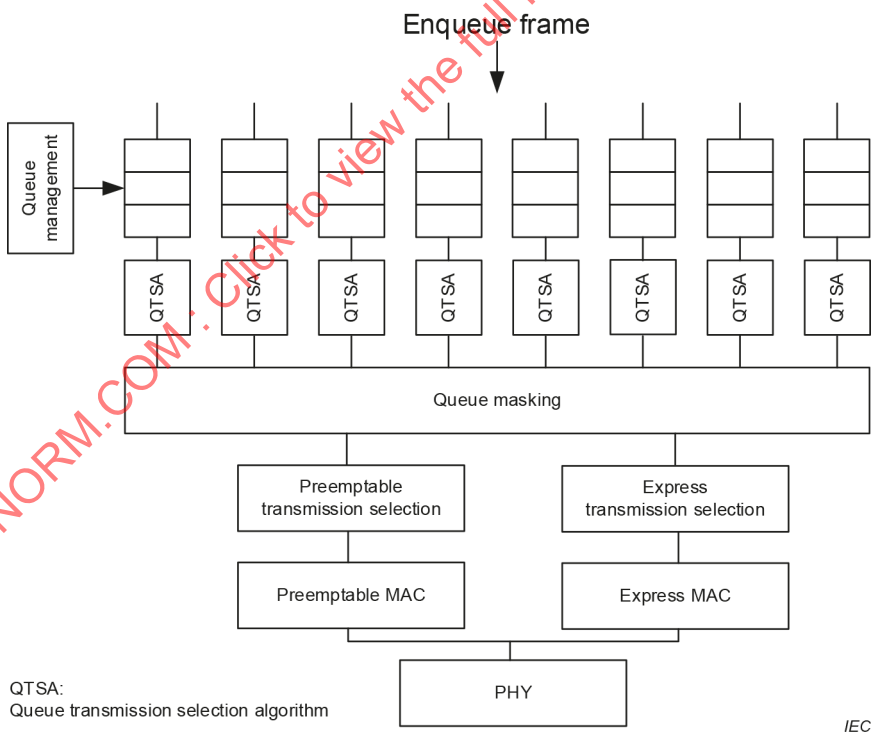


Figure 134 – Transmit – one port of a bridge

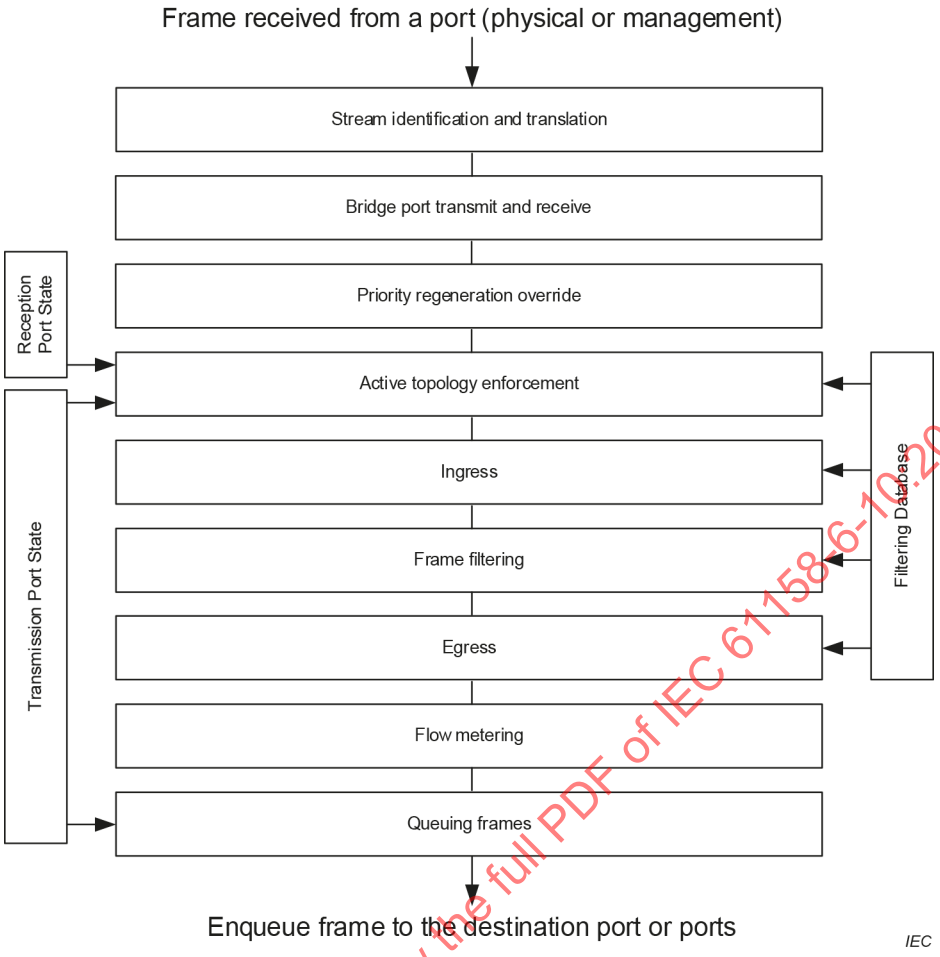


Figure 135 – Forwarding process – bridge

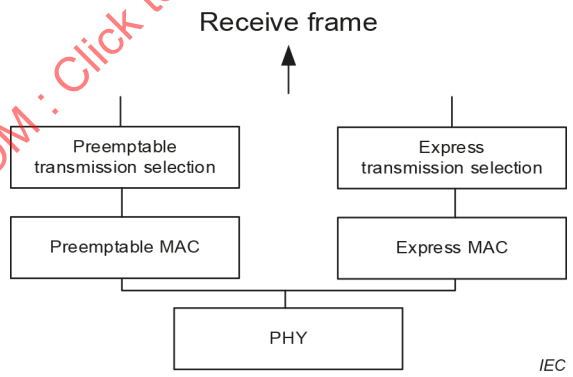


Figure 136 – Receive – one port of a bridge

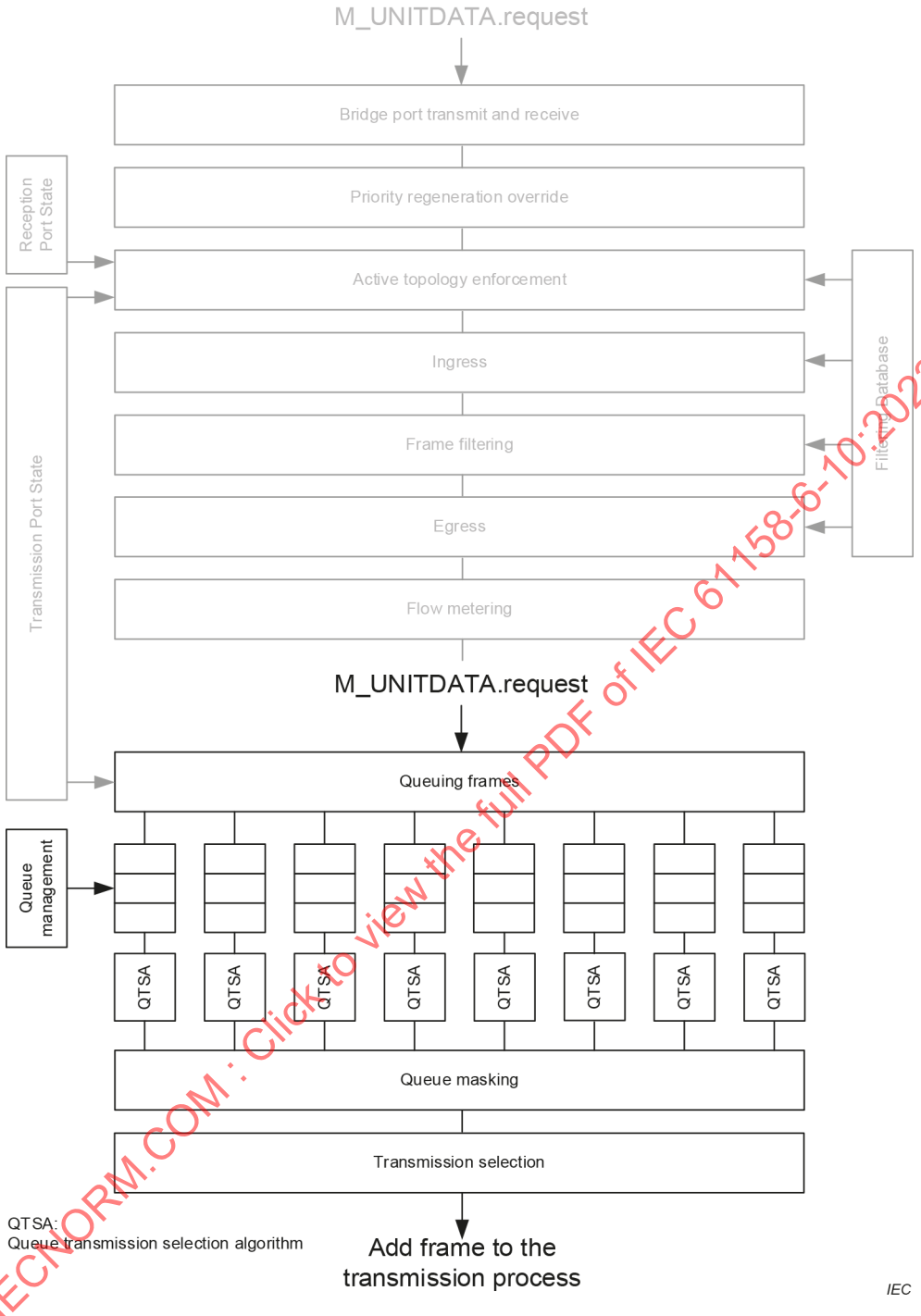


Figure 137 – Transmit – Management port

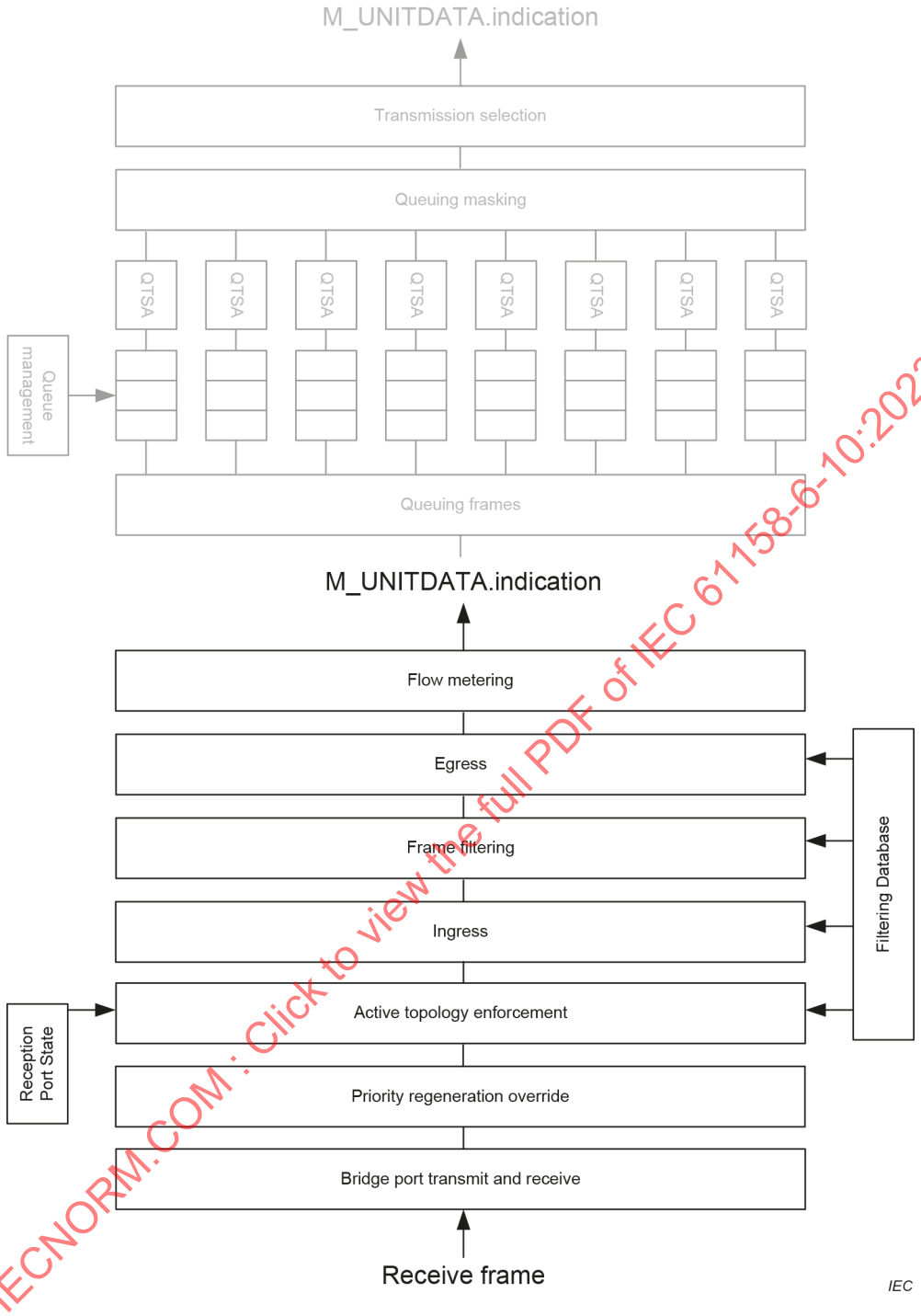


Figure 138 – Receive – Management port

4.12.5 Bridged end station

4.12.5.1 General

Figure 139 and Figure 140, as more detailed examples for Figure 118, show the device model of a bridged end station using a two port integrated bridge as an example.

Figure 141 shows the different reference points and their relation to each other.

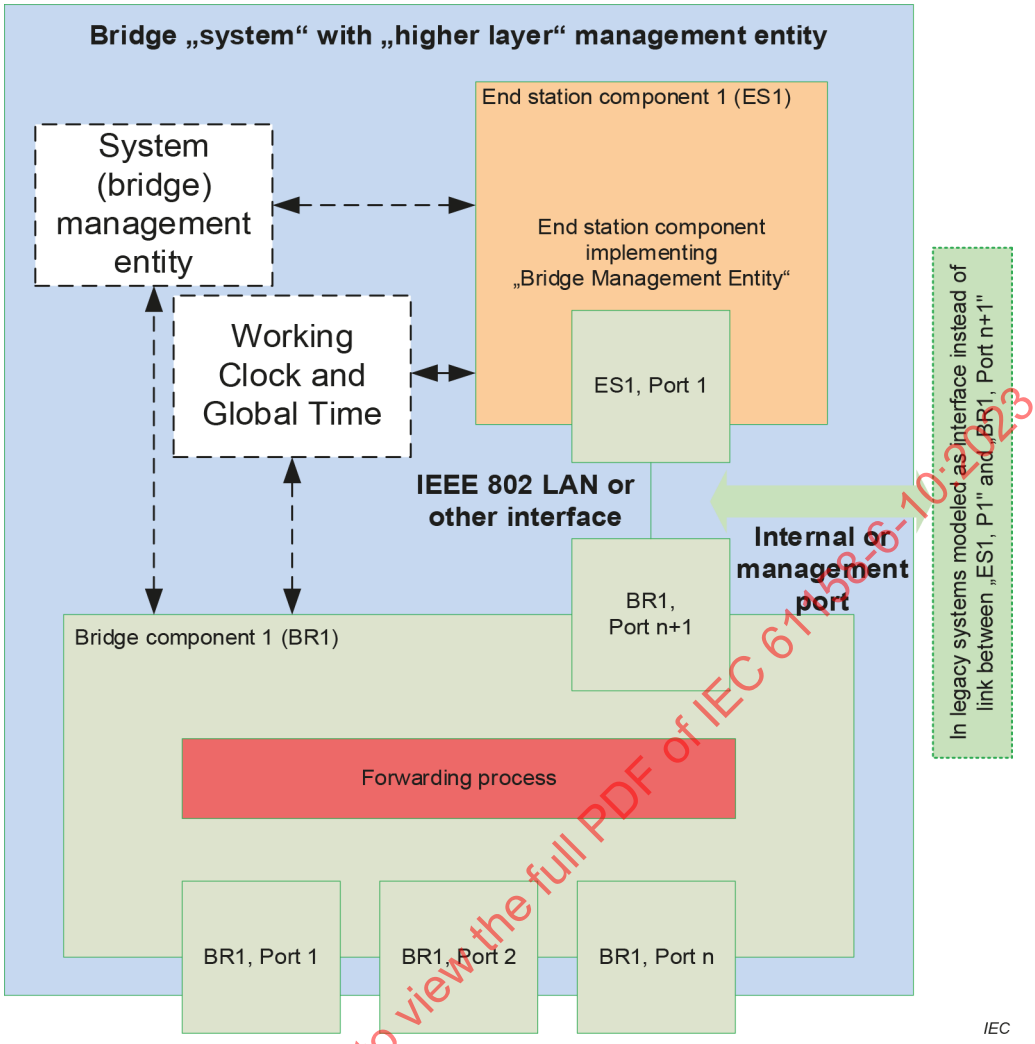


Figure 139 – Bridged end station

Figure 139 shows the limitations of a bridged end station model which doesn't cover the internal bridge and end station ports. This previously used model does not allow to differ between "BR1, Port n+1", "ES1, Port 1" and the Ethernet interface.

RT_CLASS_3:
Models the alias "BR1, Port 0" for "BR1, Port n +1". This alias is used as address for the PDIRData.

Other:
Only the Ethernet interface can be addressed and thus, parameters for "BR1, Port n+1", "ES1, Port 1" need to be derived from the Ethernet interface parameters.

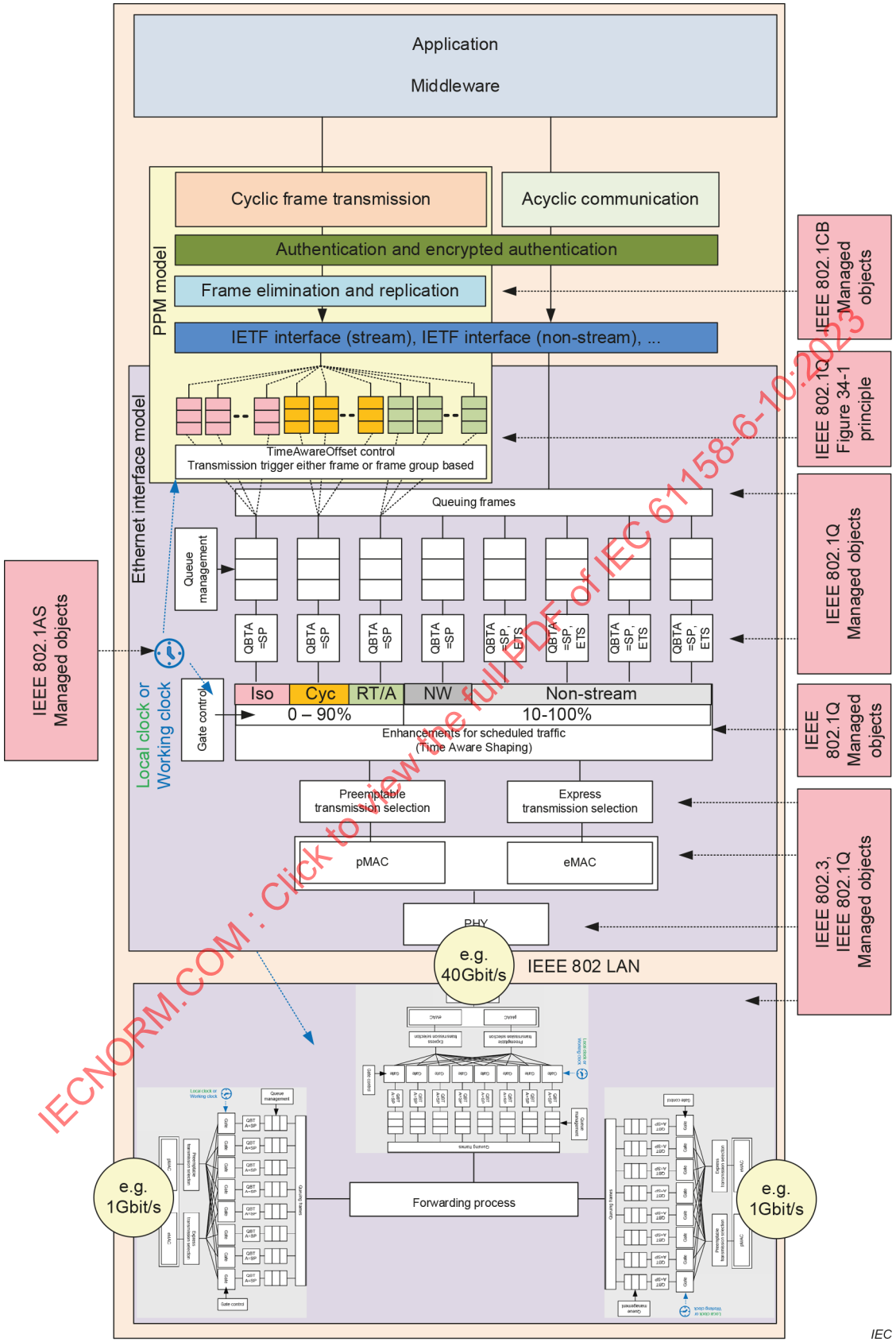


Figure 140 – Bridged end station interface model with IEEE alignment

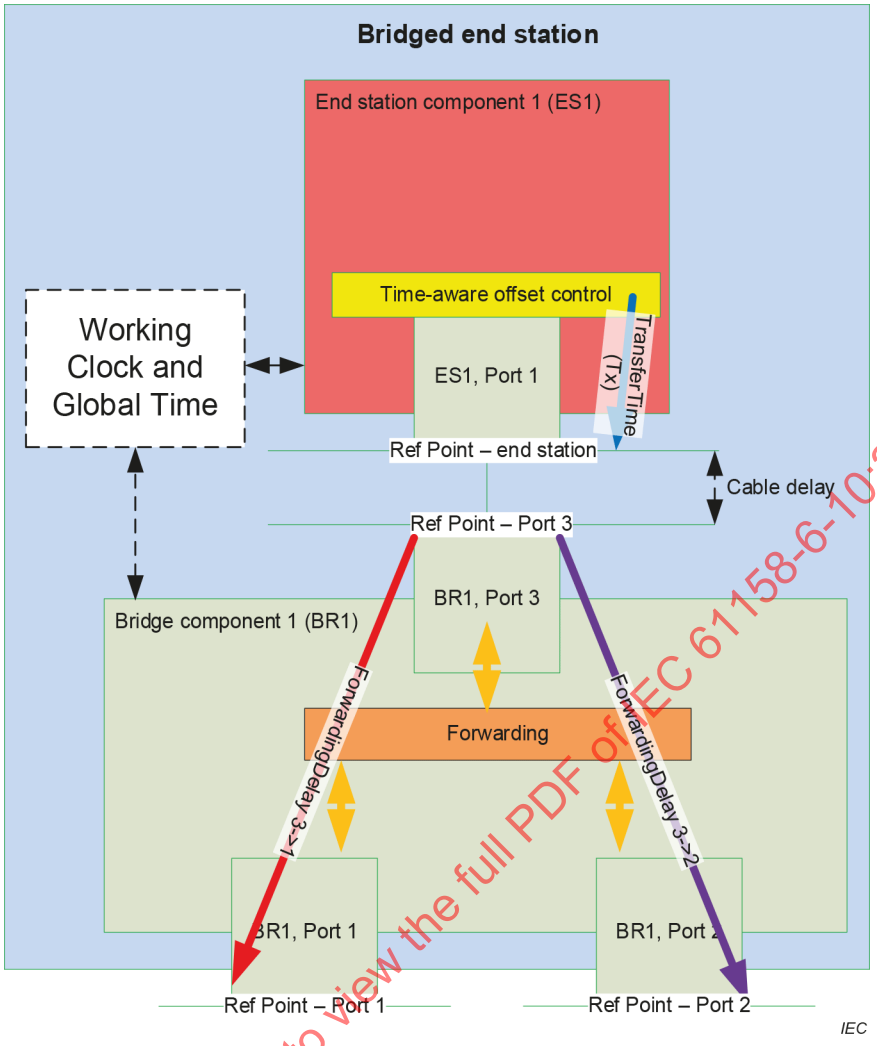


Figure 141 – Bridged end station system reference planes

4.12.5.2 Send List Control or time-aware offset control

The Ethernet interface is aware of the SendClock and ReductionRatio concept, even if the network, in case of Scheduled Traffic, only maintains one SendClock driven Scheduled Traffic window as shown in Figure 142. The same principle is valid if Preemption is used.

It provides at the beginning of the SendClock first the High+HighRed packets, second the Low+LowRed, third RT_CLASS_X (without RT_CLASS_3; only RT_CLASS_1 “Green”) and last the NRT packets (keeping the load limit) as a burst to the network.

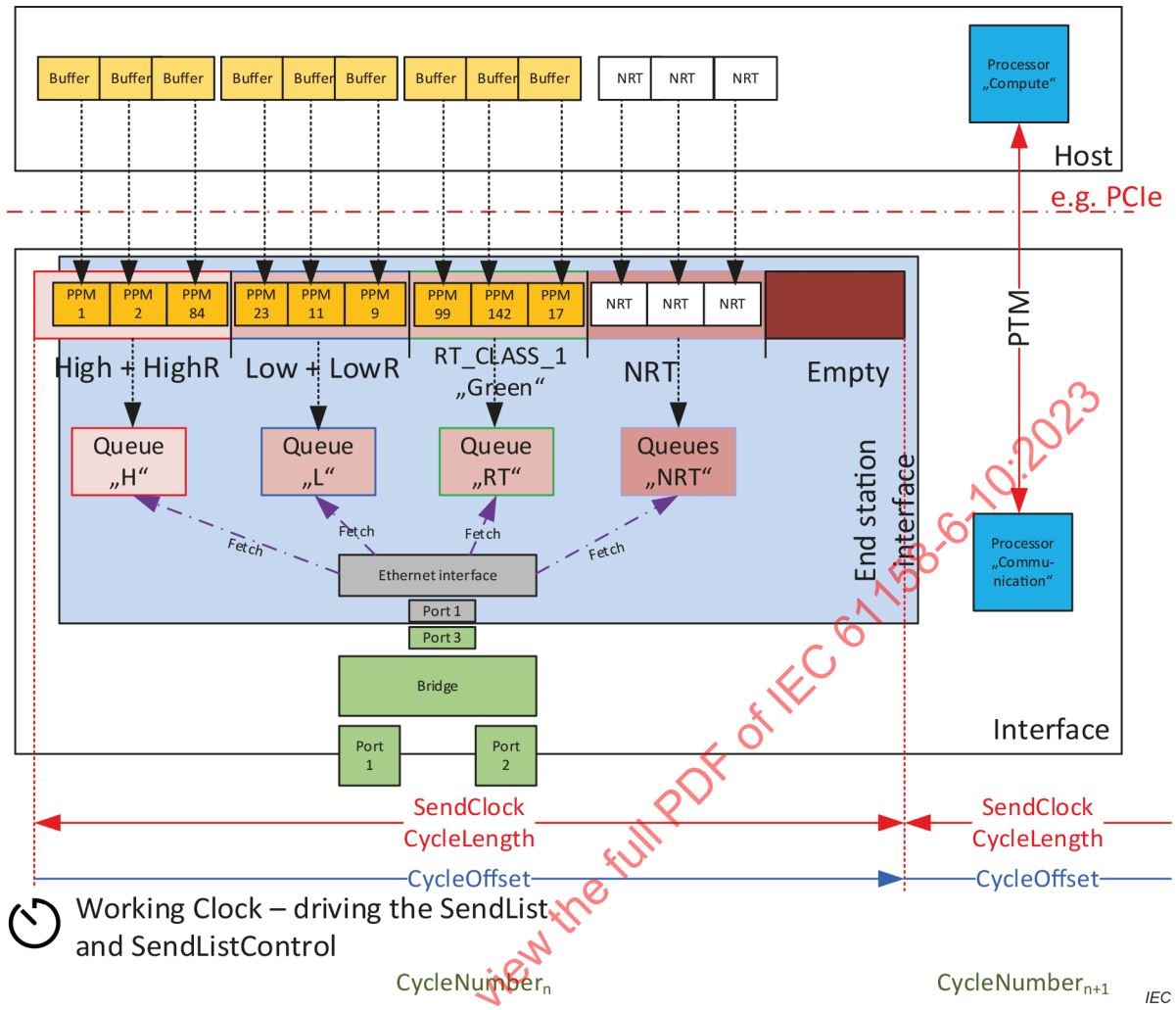


Figure 142 – Send List principle

The SendClock is created from the WorkingClock according to Formula (50) and Formula (51).

$$\text{CycleNumber} = \text{WorkingClock } \textit{div} \text{ CycleLength} \quad (50)$$

where

CycleNumber	is the integer result of the division
WorkingClock	is the actual value of the synchronized/free running WorkingClock
CycleLength	is the length of the used SendClock/Cycle

$$\text{CycleOffset} = \text{WorkingClock } \textit{mod} \text{ CycleLength} \quad (51)$$

where

CycleOffset	is the remainder result of the division
WorkingClock	is the actual value of the synchronized/free running WorkingClock
CycleLength	is the length of the used SendClock/Cycle

With each change of the CycleNumber, the 32 bit @ 1 ns counter for the CycleOffset is restarted at zero and used for the CycleOffset. Any change to the WorkingClock during the cycle has no effect on the CycleOffset if the change does not lead to a new cycle.

Even if the synchronization for the WorkingClock is lost, the cycle needs to be kept, but the Scheduled Traffic Window is no longer valid/active.

Figure 143 shows the fallback for the Working Clock in case synchronization state change from unsynchronized to synchronized. Both the application and the interface need a WorkingClock which is always valid (see 4.5) to maintain the SendClock/GatingCycle. Thus, a different clock, e.g. WorkingClock* is needed during synchronization. The application decides at which CycleNumber this intermediate timer is loaded into the WorkingClock and used.

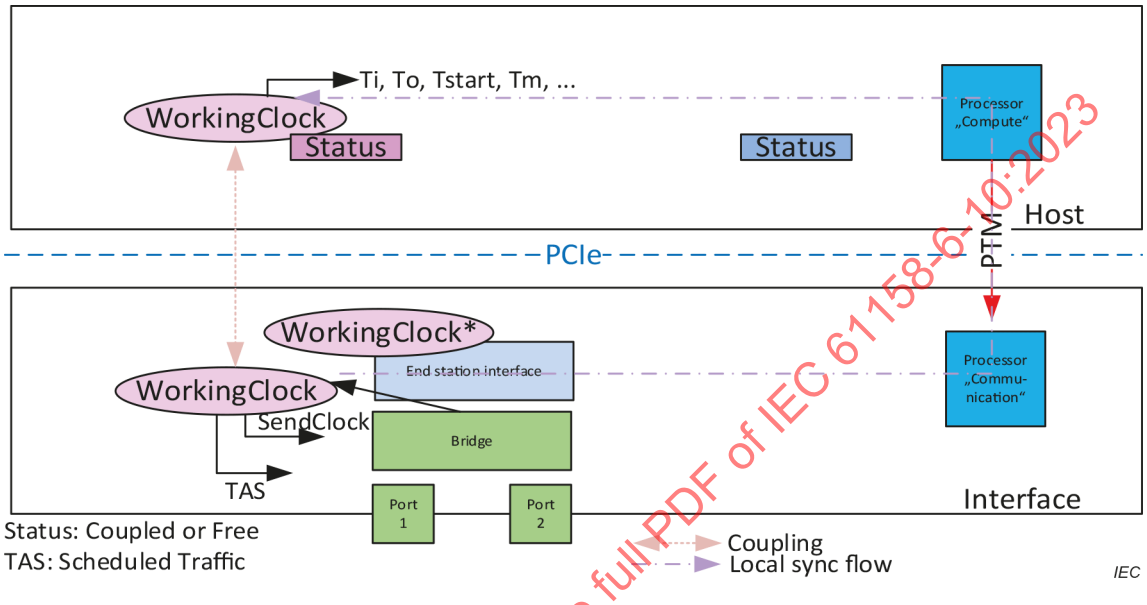


Figure 143 – Fallback in case of sync loss / resync for WorkingClock

4.12.5.3 Proprietary interfaces between end station and bridge component

IEEE Std 802.1AS and IEEE Std 802.1Q define the reference point for Working Clock at the MDI. Implementations as shown in Figure 144 suffer from port specific delays as shown in Figure 145 e.g., due to different data rates / link speeds require a port specific internal adjustment of the time-aware offset control/SendListControl.

Figure 144 shows proprietary interfaces between the end station to implement a time-aware offset control for time-aware streams. A time-aware offset control directly accesses the queues at the egress port of the bridge component using dedicated bridge resources.

Implementations according to Figure 144 lead to different reference planes for time-aware offset control as shown in Figure 145. These different reference planes, internal and external, need to be aligned for network access.

The time-aware offset control may forward time-aware streams using a proprietary interface as shown Figure 144 to the egress port of an integrated bridge. Thus, the egress port needs to handle both the frames from the forwarding process and the frames from the time-aware offset control. Figure 146 shows the basic resource model for this case.

If all time-aware streams from the time-aware offset control are forwarded immediately to the egress port, a dedicated bridge resource pool, independent from the forwarding bridge resources, is needed to avoid congestion loss during the forwarding process.

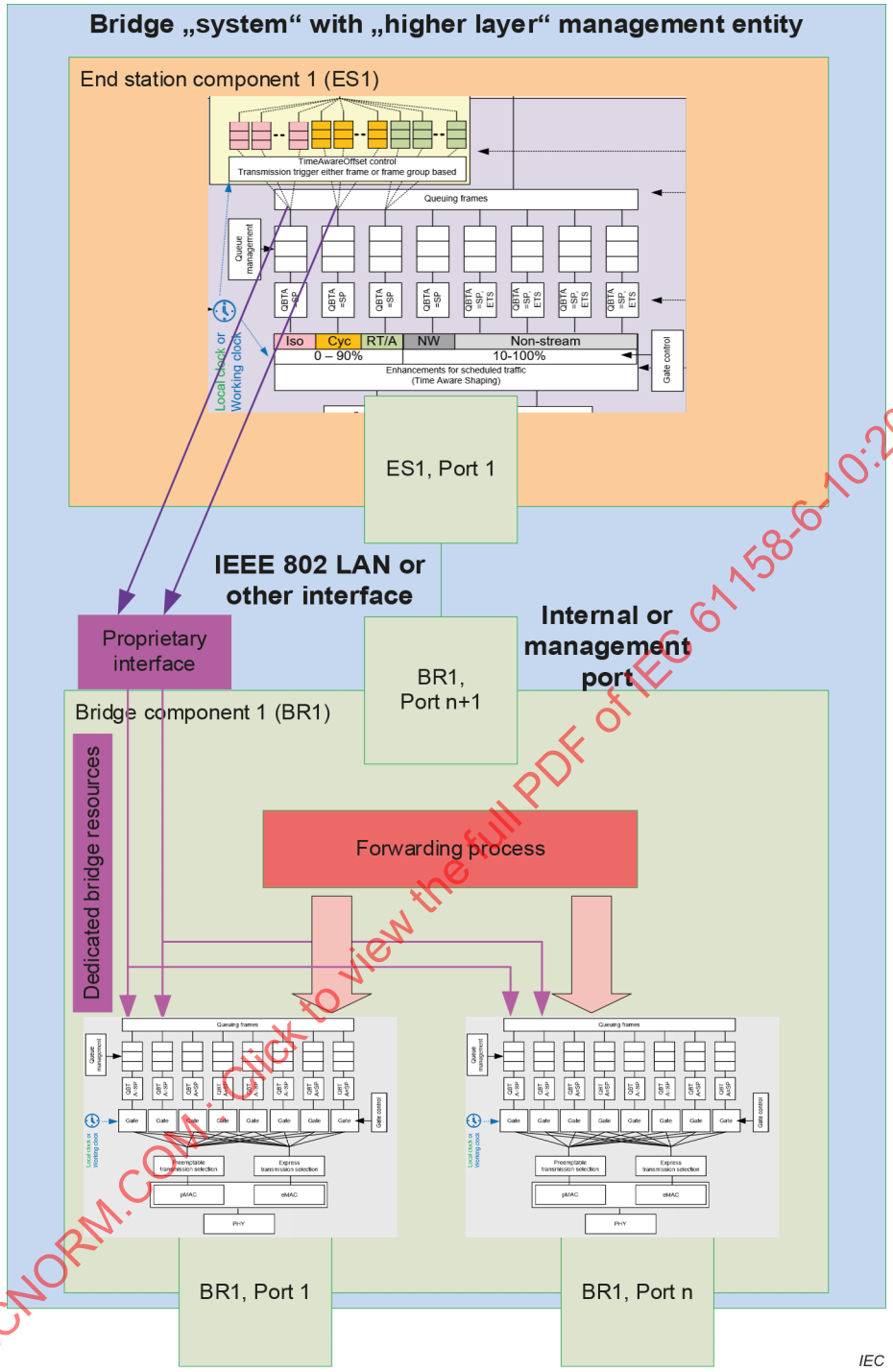


Figure 144 – Bridged end station with proprietary interfaces

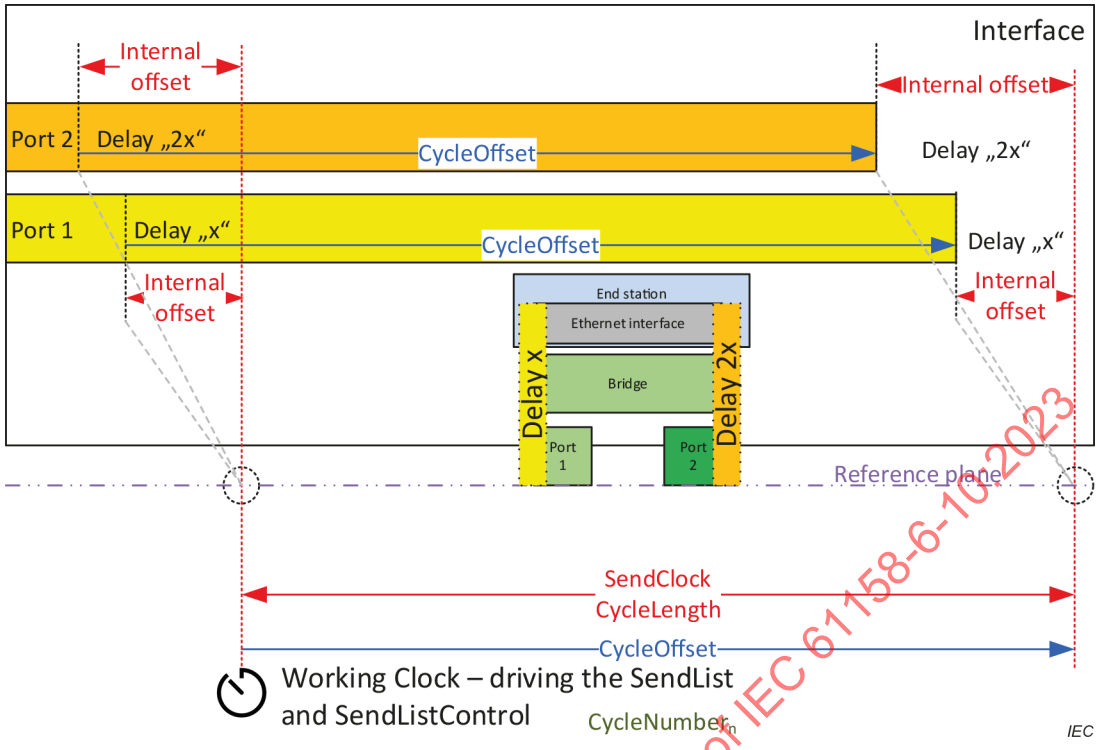


Figure 145 – Internal vs. external reference plane

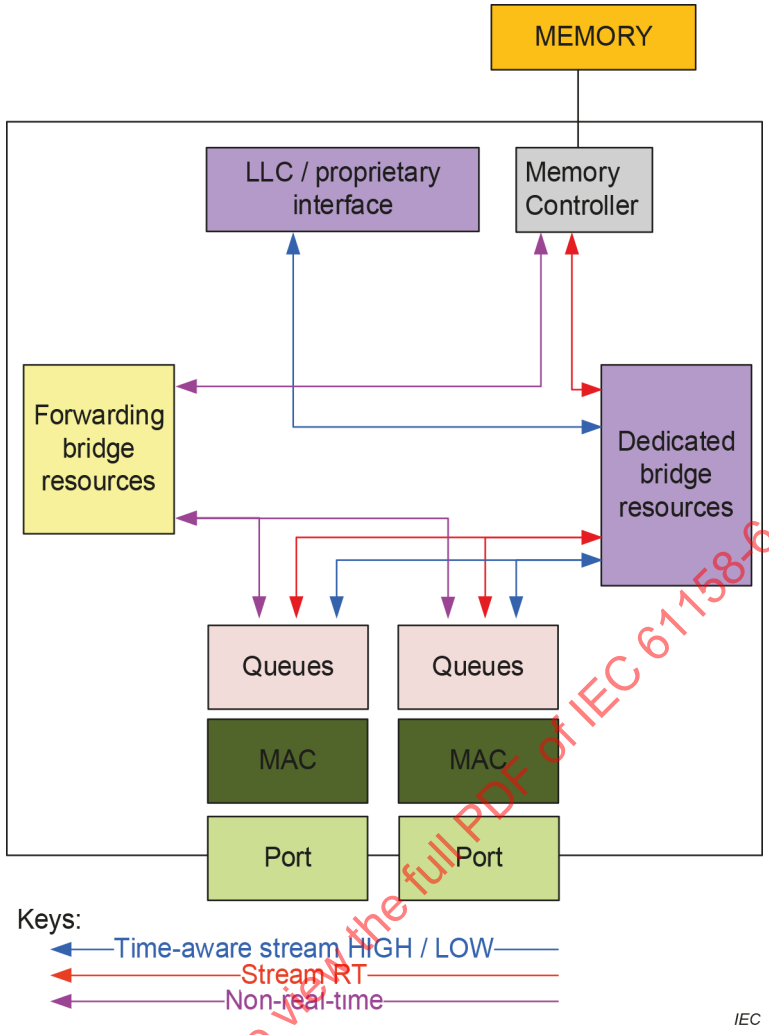


Figure 146 – Forwarding bridge resources vs. dedicated bridge resources

4.12.5.4 Examples

Figure 139, Figure 147 and Figure 148 show examples for bridged end stations. Figure 147 and Figure 148 show more advanced bridged end stations which contain multiple bridge and end station components.

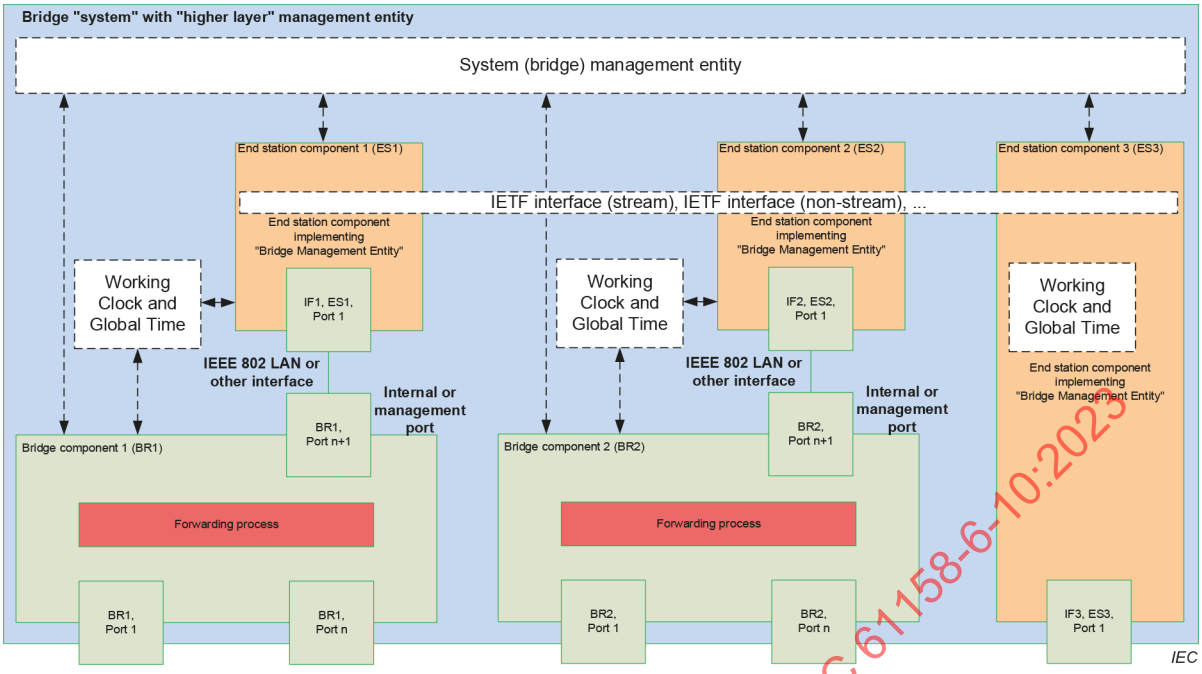


Figure 147 – Bridged end station with multiple entities – one end station per bridge component

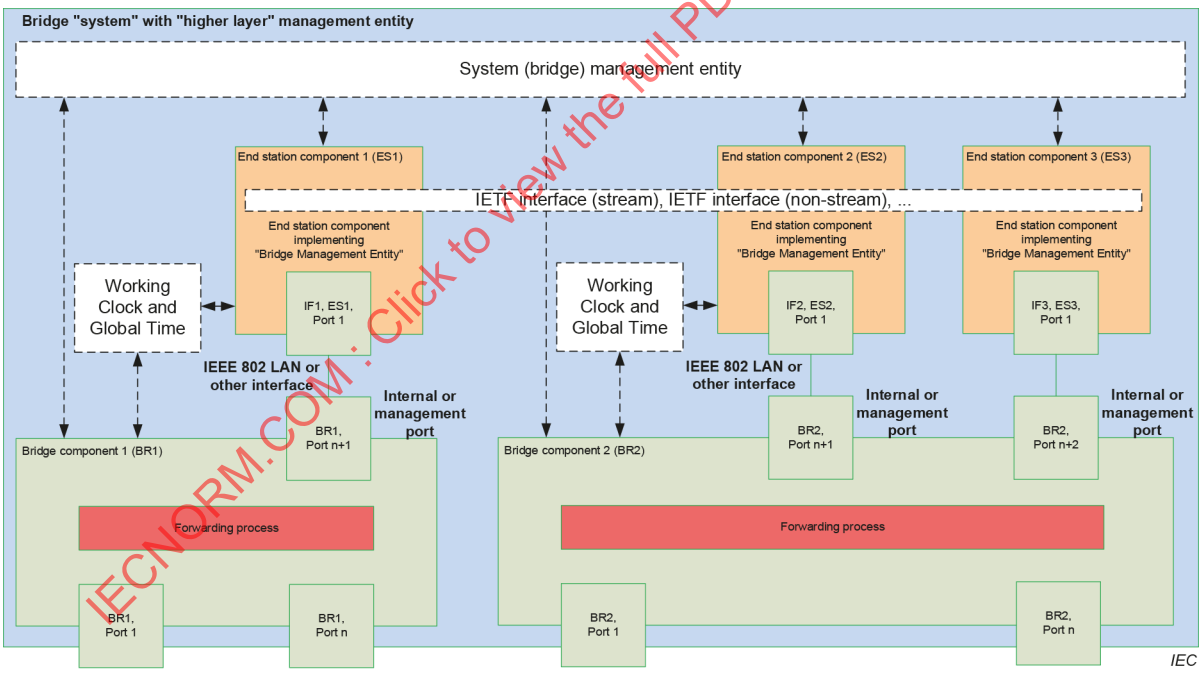


Figure 148 – Bridged end station with multiple entities – multiple end station per bridge component

4.12.6 Q port state machine

4.12.6.1 General

This document defines self-protection mechanisms for a NME domain. These mechanisms react on port base to the information received from the neighboring port. Additional, local available configuration information is used.

Thus, a port state machine is used to automatically configure IEEE Std 802.1AS, IEEE Std 802.1CB, and IEEE Std 802.1Q defined features to protect the NME domain.

4.12.6.2 Primitive definitions

4.12.6.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by IEEE Std 802.1Q Port State Machine (QPSM) are described in the service definition and shown in Table 445.

Table 445 – Remote primitives issued or received by QPSM

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.12.6.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by QPSM are described in the service definition and shown in Table 446.

Table 446 – Local primitives issued or received by QPSM

Primitive	Source	Destination	Associated parameters	Functions
Remote_Systems-_Data_Change_ind	LLDP	QPSM	Local Port ID, List of Neighbors	The LLDP informs about any neighborhood changes.
MAU_Type_Change_ind	IEEE Std 802.3	QPSM	Local Port ID, Local MAU Type, Local MAU Type Extension, Local Link Status	The IEEE Std 802.3 informs about any MAUType changes.
Port_State_Change_ind	IEEE Std 802.1Q	QPSM	Local Port ID, Local PortState Local BoundaryState	The IEEE Std 802.1Q informs about any port state changes.
SyncStateChange_ind	IEEE Std 802.1AS	QPSM	Sync State	The IEEE Std 802.1AS informs about the synchronization state.
SetPortState_req	QPSM	IEEE Std 802.1Q	Local Port ID Local PortState Local BoundaryState	Set the local port state of the given port according to the truth table.

4.12.6.3 State transition diagram

The state transition diagram of the QPSM is shown in Figure 151.

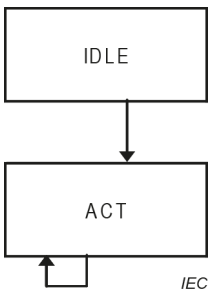


Figure 149 – State transition diagram of QPSM

States of the QPSM are:

IDLE	Idle state
ACT	Active state

4.12.6.4 State machine description

The IEEE Std 802.1Q Port State Machine (QPSM) arranges the port states of a time aware system. It relies on the information of the assigned network policy and information about the neighborhood which shall be part of the path and stream establishment. The default port state is "Boundary to IEEE Std 802.1Q device".

Additionally, it handles the enabling or disabling of the stream VLAN based forwarding.

The QPSM is only started in case this system is configured as time aware system.

Special case “Non time aware system:
A device, which is time aware but used in a non-time aware environment handles all ports as boundary ports which sets on egress either TCI.VID to zero or stripes the VLAN tag. On ingress it behaves according to 4.12.4.4.

4.12.6.5 QPSM state table

The QPSM state table is shown in Table 447 and shall exist for each port. The bandwidth allocation for TAS or the ingress rate limiter is independent from the QPSM port states.

Table 447 – QPSM state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	=> EvaluateTruthTable () SetPortState_req ()	ACT
2	ACT	**Remote_Systems_Data_Change_ind ()** => EvaluateTruthTable () SetPortState_req ()	ACT
3	ACT	**MAU_Type_Change_ind ()** => EvaluateTruthTable () SetPortState_req ()	ACT
4	ACT	**SyncStateChange_ind ()** => EvaluateTruthTable () SetPortState_req ()	ACT
5	ACT	**Port_State_Change_ind ()** => EvaluateTruthTable () SetPortState_req ()	ACT

4.12.6.6 Functions, Macros, Timers and Variables

Table 448 contains the functions, macros, timers, and variables used by the QPSM and their arguments and their descriptions.

Table 448 – Functions, Macros, Timers and Variables used by the QPSM

Name	Type	Function/Meaning
EvaluateTruthTable	Function	This local function is used as a trigger to evaluate the corresponding truth tables.
DIFFERENCE_LOCAL_OK	Macro	This macro checks whether the absolute difference between the expected local and the real local CableDelay is less or equal than 50ns. This would indicate that the assumptions which were made during the stream establishment are not fulfilled anymore and the streams are invalid.
DIFFERENCE_-REMOTE_OK	Macro	This macro checks whether the absolute difference between the local and the remote CableDelay value is less or equal than 50ns. This would indicate an asymmetry in the network and is not usable for time-aware streams.
EXPECTED_LOCAL	Macro	This local macro delivers the locally stored information which is expected to be achieved from local. The expected values are extracted from the parameterization (e.g. CIMNetConfExpectedNetworkAttributes record) which shall contain all needed data.
EXPECTED_REMOTE	Macro	This local macro delivers the locally stored information which is expected to be delivered from remote. The expected values are extracted from the parameterization (e.g. CIMNetConfExpectedNetworkAttributes record) which shall contain all needed data.
PREEMPTION_-CONFIGURED	Macro	This local macro returns true if preemption shall be used at this port, i.e. at least one queue is set to preemptible in NMEDomainQueueConfig.PreemptionMode in record CIMNetConfDataAdjust.
PREEMPTION_PARAM_VALID	Macro	This local macro returns true if the value "preemption capability support" is equal to 1 and "additional fragment size" is equal to 0. The parameters shall be filtered from an LLDP_8023_AdditionalEthernetCapabilities TLV.
IS_PROFINET_CHECK	Macro	REMOTE_EXISTS == TRUE AND REMOTE_LLDP_AVAILABLE(LLDP_PNIO_CHASSIS_MAC) == TRUE
IS_DOMAIN_CHECK	Macro	REMOTE_LLDP_AVAILABLE(LLDP_PNIO_NMEDomain) == TRUE AND REMOTE(LLDP_PNIO_NMEDomain) != ZERO_UUID AND LOCAL(LLDP_PNIO_NMEDomain) != ZERO_UUID
LOCAL	Macro	This local macro delivers the local known information.
MAUTYPE	Macro	LOCAL(MAUType) == EXPECTED_LOCAL(MAUType)
NEIGHBOR	Macro	EXPECTED_REMOTE(LLDP_PortID) == REMOTE(LLDP_PortID) AND DIFFERENCE_REMOTE_OK(LOCAL(CableDelay), REMOTE(CableDelay)) == TRUE AND (PREEMPTION_CONFIGURED== FALSE OR PREEMPTION_PARAM_VALID(REMOTE(LLDP_8023_AdditionalEthernet Capabilities)))
NME_CONFIG_CHECK	Macro	LOCAL(LLDP_NMENameUUID) == REMOTE(LLDP_NMENameUUID)) AND LOCAL(LLDP_NMENameUUID) != ZERO_UUID AND LOCAL(LLDP_NMEParameterUUID) != ZERO_UUID AND LOCAL(LLDP_NMEParameterUUID) == REMOTE(LLDP_NMEParameterUUID)
NME_DOMAIN_CHECK	Macro	LOCAL(LLDP_PNIO_NMEDomainUUID) == REMOTE(LLDP_PNIO_NMEDomainUUID)
PORT	Macro	LOCAL(NumberOfQueues) == EXPECTED_LOCAL(NumberOfQueues) AND LOCAL(PortCapabilities) == EXPECTED_LOCAL(PortCapabilities)

Name	Type	Function/Meaning
REMOTE	Macro	This local macro delivers the information received from remote. In case no such value was received, as it is an optional feature, string values are replaced by an empty string and value types are returned as zero.
REMOTE_EXISTS	Macro	Returns TRUE if exactly one neighbor exists.
REMOTE_LLDP_-AVAILABLE	Macro	Checks whether a specific LLDP-TLV is available in the LLDP frames received from remote.

SYNC	Macro	(
		DIFFERENCE_LOCAL_OK(LOCAL(CableDelay),
		EXPECTED_LOCAL(CableDelay)) == TRUE
		AND
		TAS_ENABLED == FALSE
		)
		OR
		(
		DIFFERENCE_LOCAL_OK(LOCAL(CableDelay),
		EXPECTED_LOCAL(CableDelay)) == TRUE
		AND
		TAS_ENABLED == TRUE
		AND
		// physical
		LOCAL(IN_SYNC) == TRUE
		AND
		// logical
		LOCAL(TimeDomainMasterIdentity) != 0
		AND
		LOCAL(TimeDomainMasterIdentity) ==
		REMOTE(TimeDomainMasterIdentity)
		AND
		LOCAL(TimeDomainNumber) == REMOTE(TimeDomainNumber)
		)
		
		LOCAL(IN_SYNC) returns TRUE if the local IEEE Std 802.1AS-2020 working clock sync master or sync slave is synchronized.
		The TimeDomainXXX terms only refer to the working clock entities. (Redundant and non-redundant).

ZERO_UUID	Macro	Represents an UUID with all octets set to 0
TAS_ENABLED	Macro	Returns TRUE if configured with CIMNetConfDataAdjust

4.12.6.7 Truth and behavior tables

Table 449, Table 450, Table 451 and Table 452 contains the truth behavior tables used by the QPSM.

CIMStationPortStatus provides the operational values for PortState and Express queue maintained by this state machine.

Table 449 – QPSM Port truth table

Input								Output
Conditions of the port								—
IS_-PROFINET_-CHECK [a]	IS_-DOMAIN_-CHECK	NME_-DOMAIN_-CHECK	NME_-CONFIG_-CHECK	NEIGH-BOR	PORT	MAUTYPE	SYNC [b]	PortState
FALSE	—	—	—	—	—	—	—	Boundary to IEEE Std 802.1Q device
TRUE	FALSE	—	—	—	—	—	—	Boundary to Non NME domain
TRUE	TRUE	FALSE	—	—	—	—	—	Boundary to Another NME domain
TRUE	TRUE	TRUE	FALSE	—	—	—	—	Split NME domain
TRUE	TRUE	TRUE	TRUE	FALSE	—	—	—	Split NME domain
TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	—	—	Split NME domain
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	—	Split NME domain
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	Split NME domain
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	Trunk port

[a] A device according to this document.

[b] SYNC is defined for the WorkingClock, configured for this NME domain.

Table 450 – QPSM Port ingress behavior

PortState	Unknown TCI.VID	Stream TCI.VID	Priority tagged / untagged	Non-stream TCI.VID	Other non-stream TCI.VID
Boundary to IEEE Std 802.1Q device	Drop	Drop	Forward [a]	Drop	Forward [b]
Boundary to Non NME domain	Drop	Drop	Forward [a]	Drop	Forward [b]
Boundary to Another NME domain	Drop	Forward [c]	Forward [a]	Drop	Forward [b]
Split NME domain	Drop	Drop	Drop	Forward	Forward
Trunk port	Drop	Forward	Drop	Forward	Forward

[a] Priority remapping according to 4.12.4.4.

[b] Forwarded if non-stream VIDs are known by a valid NME configuration.

[c] Forwarded if an active stream translation (IEEE Std 802.1CB) is configured for interdomain stream.

Table 451 – QPSM Port egress behavior

PortState	Unknown TCI.VID	Stream TCI.VID	Priority tagged / untagged	Non-stream TCI.VID	Other non-stream TCI.VID
Boundary to IEEE Std 802.1Q device	—	Drop	—	Forward [a]	Forward [b]
Boundary to Non NME domain	—	Drop	—	Forward [a]	Forward [b]
Boundary to Another NME domain	—	Forward [c]	—	Forward [a]	Forward [b]
Split NME domain	—	Drop	—	Forward	Forward
Trunk port	—	Forward	—	Forward	Forward

[a] Detach VLAN tag or set TCI.VID to zero.

[b] Forwarded if non-stream VIDs are known by a valid NME configuration.

[c] Forwarded if an active stream translation (IEEE Std 802.1CB) is configured for interdomain stream.

Table 452 – QPSM Port enable and disable behavior

PortState	Preemption	TAS [a]
Boundary to IEEE Std 802.1Q device	LLDP negotiated	DISABLED
Boundary to Non NME domain	LLDP negotiated	DISABLED
Boundary to Another NME domain	LLDP negotiated	DISABLED
Split NME domain	LLDP negotiated	DISABLED
Trunk port	LLDP negotiated	ENABLED

[a] Enabled only if TAS is configured

4.12.7 Pruning port state machine

4.12.7.1 Primitive definitions

4.12.7.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Pruning Port State Machine (PPSM) are described in the service definition and shown in Table 453.

Table 453 – Remote primitives issued or received by PPSM

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.12.7.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by PPSM are described in the service definition and shown in Table 454.

Table 454 – Local primitives issued or received by PPSM

Primitive	Source	Destination	Associated parameters	Functions
Remote_Systems_Data_ Change_ind	LLDP	PPSM	Local Port ID, List of Neighbors	The LLDP informs about any neighborhood changes.
Create_Static_Filtering_ Entry_req	PPSM	FDB	Local Port ID	Add the static entry for forwarding identity to the FDB.
Update_Static_Filtering_ Entry_req	PPSM	FDB	Local Port ID, Multicast MAC address	Add the MulticastMACAdd as a static entry to the FDB to forward filterable identities. Remove the added MulticastMACAdd from the FDB.

4.12.7.2 State transition diagram

The state transition diagram of the PPSM is shown in Figure 150.

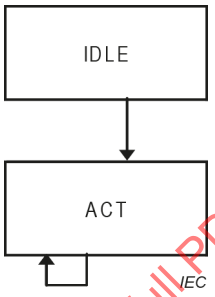


Figure 150 – State transition diagram of PPSM

States of the PPSM are:

IDLE	Idle state
ACT	Active state

4.12.7.3 State machine description

The pruning port state machine (PPSM) is using the information provided by LLDP to control DCP multicast entries in the FDB of a bridge. It updates the entries in the FDB to control the forwarding rules for the bridge port it is responsible for. Thus, it exists only at a bridge port.

The PPSM is started on power-on. See Table 17 for the used multicast MAC addresses.

Information about the neighbor, whether it is a bridge or an end station, is used together with Formula (4) and Table 20 to update the DCP multicast entries in the FDB to support multicast pruning for the DCP Identify service.

4.12.7.4 PPSM state table

The PPSM state table is shown in Table 455 and shall exist for each port. The bandwidth allocation for TAS or the ingress rate limiter is independent from the PPSM port states.

Table 455 – PPSM state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**Power-on** => Create_Static_Filtering_Entry_req (Port)	ACT
2	ACT	**Remote_Systems_Data_Change_ind ()** /EvaluateTruthTable () = DEFAULT => Update_Static_Filtering_Entry_req (Port)	ACT
3	ACT	**Remote_Systems_Data_Change_ind ()** /EvaluateTruthTable () = OPTIMIZED => Calculate_MulticastMACAdd () Update_Static_Filtering_Entry_req (Port, MulticastMACAdd)	ACT

4.12.7.5 Functions, Macros, Timers and Variables

Table 456 contains the functions, macros, timers, and variables used by the PPSM and their arguments and their descriptions.

Table 456 – Functions, Macros, Timers and Variables used by the PPSM

Name	Type	Function/Meaning
EvaluateTruthTable	Function	This local function is used as a trigger to evaluate the corresponding truth tables.
Calculate_MulticastMACAdd	Function	This local function calculates the DCP_MulticastMACAdd assigned to the received NameOfStation. See DCP_MulticastMACAdd range for filterable Identify.

4.12.7.6 Truth and behavior tables

Table 457 contains the truth behavior tables used by the PPSM.

Table 457 – PPSM truth table

Input		Output
Neighbor is end station	**NameOfStation available**	**FDB entries**
FALSE	—	DEFAULT forwarding rules for multicast frames apply
TRUE	FALSE	DEFAULT forwarding rules for multicast frames apply
TRUE	TRUE	OPTIMIZED forwarding rules for multicast frames apply

4.12.8 Bridge extensions

Apart from IEEE Std 802.1Q behavior, this protocol specification defines the following protocol machines to provide special forwarding actions:

– RT_CLASS_3 Forwarding Protocol Machine (RED_RELAY)
– DFP Forwarding Protocol Machine (DFP_RELAY)
– Time Synchronization Forwarding Protocol Machine (SYNC_RELAY)
– Queue handler Protocol Machine (QueueHandler)
– Multiplex MAC Protocol Machine (MUX)

– Demultiplex MAC Protocol Machine (DEMUX)

Additionally the learning mechanisms of IEEE Std 802.1Q shall be disabled according to Table 411, Table 412 and Table 413. As an optimization a previous learned entry of the FDB may be refreshed.

The routing mechanisms of IEEE Std 802.1Q shall be temporary disabled within the RT_CLASS_3 phase. In this case, the routing of RT_CLASS_3 frames shall be done according to the attributes defined by the appropriate ASE. The values of the attributes define a special routing table, which is used.

4.12.9 FAL Service Protocol Machines

There is no FAL Service Protocol Machine (FSPM) defined for this Protocol.

4.12.10 Application Relation Protocol Machines

There is no Application Relation Protocol Machine (ARPM) defined for this Protocol.

4.12.11 DLL Mapping Protocol Machines

4.12.11.1 QueueHandler

The QueueHandler manages the priority queues of a port. It offers service primitives for enqueuing and dequeuing of frames. Additionally, information on the content of a queue is available.

4.12.11.2 Forwarding Protocol Machine

4.12.11.2.1 Primitive definitions

####### 4.12.11.2.1.1 General

The service primitives including their associated parameters issued by MAC_RELAY user and received by MAC_RELAY or vice versa are described in the MAC bridges ASE in the service definition.

####### 4.12.11.2.1.2 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by MAC_RELAY are described in the service definition and shown in Table 458.

Table 458 – Remote primitives issued or received by MAC_RELAY

Primitive	Source	Destination	Associated parameters	Functions
MAC_RELAY_Data.req	LMPM	MAC_RELAY	Frame	—
MAC_RELAY_Data.cnf	LMPM	MAC_RELAY	Frame, Status	—
MAC_RELAY_Data.ind	MAC_RELAY	LMPM	Port, Frame, Timestamp	—
DEMUX_MAC_RELAY.ind	DEMUX	MAC_RELAY	Port, Frame	—
QueueHandler.req	MAC_RELAY	QueueHandler	Queue, Frame	—

4.12.11.2.1.3 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by MAC_RELAY are described in the service definition and shown in Table 459.

Table 459 – Local primitives issued or received by MAC_RELAY

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.12.11.2.2 State transition diagram

The state transition diagram of the MAC_RELAY shall be according to IEEE Std 802.1Q.

4.12.11.2.3 State machine description

The state machine of the MAC_RELAY shall be according to IEEE Std 802.1Q.

4.12.11.2.4 MAC_RELAY state table

The state table shall be according to IEEE Std 802.1Q.

4.12.11.2.5 Functions, Macros, Timers and Variables

Table 460 contains the functions, macros, timers, and variables used by the MAC_RELAY and their arguments and their descriptions.

Table 460 – Functions, Macros, Timers and Variables used by the MAC_RELAY

Name	Type	Function/Meaning
—	—	—

4.12.11.3 RT_CLASS_3 port state machine

4.12.11.3.1 General

This part of the specification defines the utilization of the IEEE Std 802.1Q.

It includes extensions for RT_CLASS_3 phase regarding forwarding of frames.

The routing mechanisms of IEEE Std 802.1Q shall be temporarily disabled within the RT_CLASS_3 phase. In this case, the routing of RT_CLASS_3 frames shall be done according to the attributes defined by the appropriate ASE. The values of the attributes define a special routing table, which is used.

4.12.11.3.2 Primitive definitions

4.12.11.3.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Realtime Class 3 Port State Machine (RTC3PSM) are described in the service definition and shown in Table 461.

Table 461 – Remote primitives issued or received by RTC3PSM

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.12.11.3.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by RTC3PSM are described in the service definition and shown in Table 462.

Table 462 – Local primitives issued or received by RTC3PSM

Primitive	Source	Destination	Associated parameters	Functions
Remote_Systems_Data_Change_ind	LLDP	RTC3PSM	Local Port ID, List of Neighbors	—
MAU_Type_Change_ind	IEEE Std 802.3	RTC3PSM	Local Port ID, Local MAU Type, Local MAU Type Extension, Local Link Status	—
Port_State_Change_ind	IEEE Std 802.1Q	RTC3PSM	Local PortState	—
SyncStateChange_ind	PTCP	RTC3PSM	Sync State	—
SetPortState	RTC3PSM	IEEE Std 802.1Q	Local PortState	—

4.12.11.3.3 State transition diagram

The state transition diagram of the RTC3PSM is shown in Figure 151.

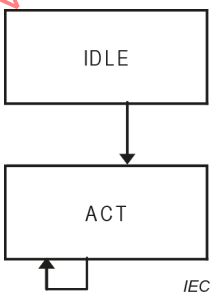


Figure 151 – State transition diagram of RTC3PSM

States of the RTC3PSM are:

IDLE	Idle state
ACT	Active state

4.12.11.3.4 State machine description

The Realtime Class 3 Port State Machine (RTC3PSM) arranges the port states of realtime class 3. It relies on the information from the engineering about the neighborhood which, provided as records during startup, shall be part of the connection establishment.

Special case “DummyMode”:
A device, only loosely coupled as end station due to missing hardware support for SYNC, shall be supported. This case is indicated by the response to LineDelay measurement and an

internally defined LineDelay := “DummyMode” value. The DIFFERENCE_FAULT(CableDelay) and the check of the LineDelay shall not return a fault for this port in this case.

4.12.11.3.5 RTC3PSM state table

The RTC3PSM state table is shown in Table 463 and shall exist for each port. The bandwidth allocation using the TXBeginEndAssignment and the RXBeginEndAssignment shall be coupled with the RTC3PSM port states.

Table 463 – RTC3PSM state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	=> EvaluateTruthTable() SetPortState ()	ACT
2	ACT	**Remote_Systems_Data_Change_ind ()** => REMOTEPORTSTATE NEIGHBOR ENGINEERING EvaluateTruthTable() SetPortState ()	ACT
3	ACT	**MAU_Type_Change_ind ()** => PORT EvaluateTruthTable() SetPortState ()	ACT
4	ACT	**SyncStateChange_ind ()** => SYNC EvaluateTruthTable() SetPortState ()	ACT
5	ACT	**Port_State_Change_ind ()** => PORT EvaluateTruthTable() SetPortState ()	ACT

4.12.11.3.6 Functions, Macros, Timers and Variables

Table 464 contains the functions, macros, timers, and variables used by the RTC3PSM and their arguments and their descriptions.

Table 464 – Functions, Macros, Timers and Variables used by the RTC3PSM

Name	Type	Function/Meaning
EvaluateTruthTable	Function	This function is used as a trigger to evaluate the corresponding truth table.
CHECK_DATA	Macro	This function delivers an information whether the local stored data is valid.
DIFFERENCE_FAULT	Macro	This function checks whether the difference between the local and remote measured CableDelay is less than 50 ns.

ENGINEERING	Macro	! (
		// physical
		EXPECTED_REMOTE(IRDataUUID) !=
		REMOTE(LLDP_IRDataUUID)
		OR
		EXPECTED_REMOTE(PTCP_SubdomainUUID) !=
		REMOTE(LLDP_PTCP_SubdomainUUID)
		OR
		if local RT_CLASS_3_Port_State != RUN:
		EXPECTED_REMOTE(PTCP_MasterMAC) !=
		REMOTE(LLDP_PTCP_MasterMAC)
		)

EXPECTED_LOCAL	Macro	This function delivers the locally stored information which is expected to be achieved from local. The expected values are extracted from the parameterization (for example PDPortDataCheck record) which shall contain all needed data.
EXPECTED_REMOTE	Macro	This function delivers the locally stored information which is expected to be delivered from remote. The expected values are extracted from the parameterization (for example PDPortDataCheck record) which shall contain all needed data.
GET_DOMAIN_BOUNDARY_VALUE	Macro	This function delivers the local information, stored in PDIRGlobalData and PDIRBeginEndData, whether this port is used as a RT_CLASS_3 port or not.
LOCAL	Macro	This function delivers the local information.
NEIGHBOR	Macro	! (// physical EXPECTED_REMOTE(LLDP_ChassisID) != REMOTE(LLDP_ChassisID) OR EXPECTED_REMOTE(LLDP_PortID) != REMOTE(LLDP_PortID) OR REMOTE(MAUType) != VALID [a] OR (EXPECTED_REMOTE(MAUType) != LOCAL(MAUType)) OR (LOCAL(LineDelay) > EXPECTED_LOCAL(LineDelay))) [a] This function returns for the parameter MAUType the value VALID, if the speed is greater or equal 100 Mbit/s and the duplex mode is full according to Table 814.

PORT	Macro	! (
		// physical
		LOCAL(MAUType) != VALID [a]
		OR
		(EXPECTED_LOCAL(MAUType) != LOCAL(MAUType))
		OR
		LOCAL(PortState) == DISCARDING
		OR
		LOCAL(LinkStatus) != UP
		OR
		// logical
		LOCAL(GET_DOMAIN_BOUNDARY_VALUE()) == TRUE
		)
		
		[a] This function returns for the parameter MAUType the value VALID, if the speed is greater or equal 100 Mbit/s and the duplex mode is full according to Table 814.

Name	Type	Function/Meaning
REMOTE	Macro	This function delivers the remote information stored locally.
REMOTEPORTSTATE	Macro	(// physical Checks if the remote RT_CLASS_3_Port_State exists and contains a value from the following list {OFF, RTCLASS3_UP, RTCLASS3_RUN})

SYNC	Macro	! (
		// physical
		LOCAL(SYNC) == FALSE [a]
		OR
		// logical
		(DIFFERENCE_FAULT(LOCAL(CableDelayLocal), REMOTE(CableDelayRemote)))
		)
		[a] This function returns TRUE if the local PTCP master or PTCP slave is synchronized.

4.12.11.3.7 Truth table

Table 465 and Table 466 contain the truth table used by the RTC3PSM.

Table 465 – Truth table for the RTC3PSM

Input						Output
Conditions of the transmit port				Remote port state	Local port state	Next local port state
PORT	NEIGHBOR	SYNC	ENGINEERING [b]	—	—	—
FALSE	—	—	—	—	—	OFF
TRUE	FALSE	—	—	—	—	OFF
TRUE	TRUE	FALSE	—	—	—	OFF
TRUE	TRUE	TRUE	FALSE	—	—	OFF
TRUE	TRUE	TRUE	TRUE	OFF	—	UP
TRUE	TRUE	TRUE	TRUE	UP	—	RUN
TRUE	TRUE	TRUE	TRUE	RUN [c]	— [a]	RUN

[a] The local state "OFF" in combination with remote port state "RUN" is not suitable, but possible as a transient state. Even in this case, the next local state should be "RUN".

[b] PDIRData and if needed PDIRSubframeData exist and are valid.

[c] RT_CLASS_3 frames should only be transmitted in this case.

Table 466 – RXBeginEndAssignment and TXBeginEndAssignment

Input		Output	
Local port state	**Remote port state**	**Local RXBeginEndAssignment**	**Local TXBeginEndAssignment**
OFF	—	OFF	OFF
UP	—	OFF	ON [a]
RUN	UP	ON	ON [a]
RUN	RUN	ON	ON [b]

[a] To protect the installation of a remote RXBeginEndAssignment the local TXBeginEndAssignment should start with the RedOrangePeriodBegin:= 0, even if the PDIRBeginEndData stated otherwise.

[b] After the installation of a remote RXBeginEndAssignment the local TXBeginEndAssignment should start with the RedOrangePeriodBegin from the PDIRBeginEndData.

4.12.11.3.8 Generating of events

The RTC3PSM shows the state changes required to establish a RT_CLASS_3 peer to peer communication. In addition, Figure 152 shows the diagnostic events, generated according to the state changes of the RTC3PSM. Table 467 defines the coding of the associated events.

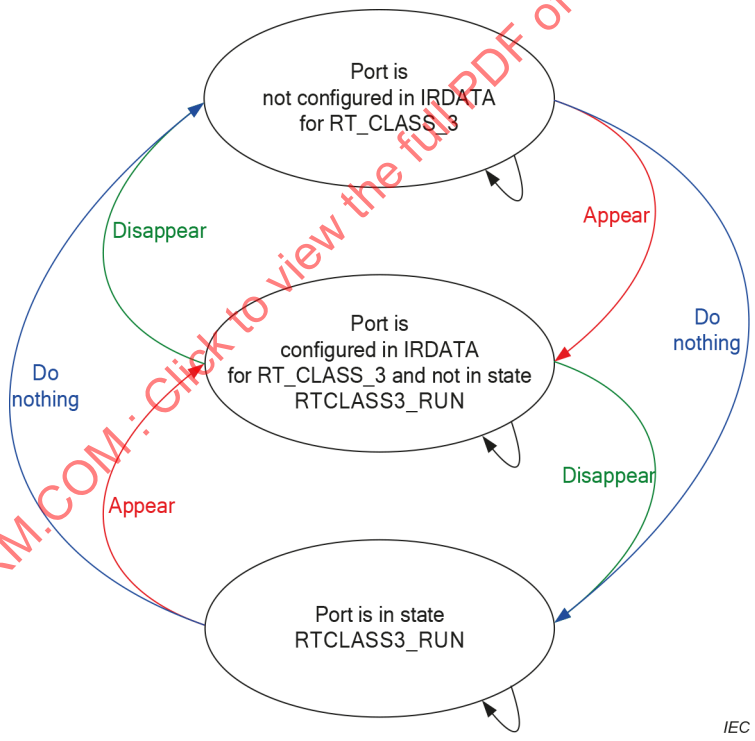


Figure 152 – State transition diagram for generating events

Table 467 – Event function table

Function name	Operations
Appear	Alarm Priority := ALARM_HIGH or ALARM_LOW Alarm Type:=Port Data Change Notification Alarm User Structure Identifier := 0x8002 ChannelNumber := 0x8000 ChannelProperties.Specifier := Appears ChannelErrorType := “Remote Mismatch” ExtChannelErrorType := “RT Class3 mismatch” Alarm Notification.req(AREP, API, Alarm Priority, Alarm Type, Slot Number, Subslot Number, Alarm Specifier, Module Ident Number, Submodule Ident Number, Alarm Item)
Disappear	Alarm Priority := ALARM_HIGH or ALARM_LOW Alarm Type:=Port Data Change Notification Alarm User Structure Identifier := 0x8002 ChannelNumber := 0x8000 ChannelProperties.Specifier := Disappears [a] ChannelErrorType := “Remote Mismatch” ExtChannelErrorType := “RT Class3 mismatch” Alarm Notification.req(AREP, API, Alarm Priority, Alarm Type, Slot Number, Subslot Number, Alarm Specifier, Module Ident Number, Submodule Ident Number, Alarm Item)
Do nothing	This state change shall not indicate an event.

[a] This value is used to illustrate the principle. The rules for ChannelProperties.Specifier apply.

4.12.11.4 RT_CLASS_3 Forwarding Protocol Machine

4.12.11.4.1 Primitive definitions

4.12.11.4.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Realtime Class 3 Forwarding Protocol Machine (RED_RELAY) are described in the service definition and shown in Table 468.

Table 468 – Remote primitives issued or received by RED_RELAY

Primitive	Source	Destination	Associated parameters	Functions
DMUX_RED_RELAY.ind	DFP_RELAY, DEMUX	RED_RELAY	Port, Tstamp, DA, SA, N_SDU, ReceivedInRED	—
RED_RELAY_Data.ind	RED_RELAY	LMPM	Port, Tstamp, DA, SA, N_SDU, ReceivedInRED	—
QueueHandler.req	RED_RELAY	QueueHandler	RED	This function puts the frame in the RED queue of the QueueHandler. The MUX dequeues the frame and conveys it to the MAC.

4.12.11.4.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by RED_RELAY are described in the service definition and shown in Table 469.

Table 469 – Local primitives issued or received by RED_RELAY

Primitive	Source	Destination	Associated parameters	Functions
RED_RELAY_Error_Ind	RED_RELAY	Local matter	CREP, ErrorCode	—
SyncStateChange_ind	PTCP	RED_RELAY	SYNC	—

4.12.11.4.2 State transition diagram

The state transition diagram of the RED_RELAY is shown in Figure 153.

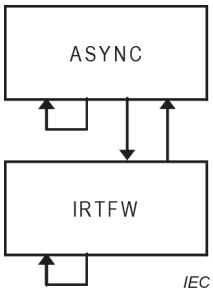


Figure 153 – State transition diagram of RED_RELAY

States of the RED_RELAY are:

ASYNC The node shall be synchronized, either locally as a sync master or remotely as a sync slave for the RED forwarding

IRTFW After the node has been synchronized, the forwarding and local reception of RED frames using the PDIRFrameData or the PDIRGlobalData and PDIRBeginEndData is done

4.12.11.4.3 State machine description

The Realtime Class 3 Forwarding Protocol Machine (RED_RELAY) conducts the forwarding and local reception of RED frames.

The RED_RELAY shall ensure that the forwarding of the RT_CLASS_3 frames is done in a timely correct way. This means that the frame shall be forwarded according to the given FrameSendOffset (see Figure 44 for the FrameSendOffset reference point), to work with all kinds of RedGuards.

Devices supporting the RedGuard with full check may generate surrogate frames (see Table 220 for the used values of APDU_Status.TransferStatus), if a RED_RELAY_Error_Ind is detected or a frame is missing.

The necessary data for the RED_RELAY is given by the PDIRFrameData or locally generated from the PDIRGlobalData.

4.12.11.4.4 RED_RELAY state table

The RED_RELAY state table is shown in Table 470.

Table 470 – RED_RELAY state table

#	Current State	Event /Condition =>Action	Next State
1	ASYNC	**SyncStateChange_ind ()** /SYNC => ignore	IRTFW
2	ASYNC	**SyncStateChange_ind ()** /!SYNC => ignore	ASYNC
3	ASYNC	**TickEvent ()** => ignore	ASYNC
4	ASYNC	**DMUX_RED_RELAY.ind ()** => ignore	ASYNC
5	IRTFW	**SyncStateChange_ind ()** /SYNC => ignore	IRTFW
6	IRTFW	**SyncStateChange_ind ()** /!SYNC => ignore	ASYNC
7	IRTFW	**TickEvent (CycleCounter, CycleOffset)** => Calculate Local_Phase as input for the RedGuard Calculate Local_FrameSendOffset as input for the RedGuard	IRTFW
8	IRTFW	**DMUX_RED_RELAY.ind ()** /RedGuard () == DISCARD, Reason == WrongTxPortState => ErrorCode := WRONG_TX_STATE RED_RELAY_Error_Ind (CREP, ErrorCode)	IRTFW
9	IRTFW	**DMUX_RED_RELAY.ind ()** /RedGuard () == DISCARD, Reason == Data length => ErrorCode := WRONG_LENGTH RED_RELAY_Error_Ind (CREP, ErrorCode)	IRTFW
10	IRTFW	**DMUX_RED_RELAY.ind ()** /RedGuard () == DISCARD, Reason == FrameID OR Reason == Receive Port OR Reason == Phase => ErrorCode := WRONG_FRAMEID RED_RELAY_Error_Ind (CREP, ErrorCode)	IRTFW
11	IRTFW	**DMUX_RED_RELAY.ind ()** /RedGuard () == DISCARD, Reason == FrameSendOffset => ErrorCode := WRONG_FrameSendOffset RED_RELAY_Error_Ind (CREP, ErrorCode)	IRTFW

12	IRTFW	**DMUX_RED_RELAY.ind ()**
		/RedGuard () == FORWARD, List of TX Ports
		=>
		StartTimer (IRT, Expected FrameSendOffset)
		For (TimerExpired (IRT), List of TX ports)
		if TX Port == 0
		RED_RELAY_Data.ind ()
		else
		QueueHandler.req ()
			IRTFW

4.12.11.4.5 Functions, Macros, Timers and Variables

Table 471 contains the functions, macros, timers, and variables used by the RED_RELAY and their arguments and their descriptions.

Table 471 – Functions, Macros, Timers and Variables used by the RED_RELAY

Name	Type	Function/Meaning
RedGuard	Function	This function checks the received frame against the local information.
StartTimer	Function	Starts the IRT Timer which controls the timely correct transmission of a frame.
TickEvent	Function	CycleCounter, CycleOffset
TimerExpired	Function	Controls the timely correct transmission of a frame.
Calculate Local_FrameSendOffset as input for the RedGuard	Macro	Calculates the input parameter Local_FrameSendOffset for the RedGuard.
Calculate Local_Phase as input for the RedGuard	Macro	Calculates the input parameter Local_Phase for the RedGuard.
IRT_Timer	Timer	Timer which is used to control the timely correct transmission of a frame by generating the FrameSendOffset event.
Expected FrameSendOffset	Variable	Expected or calculated FrameSendOffset of the frame.
Local_FrameSendOffset	Variable	This local variable contains the actual value of the FrameSendOffset driven by TickEvent.
Local_Phase	Variable	This local variable contains the actual value of the Phase driven by TickEvent.

4.12.11.4.6 RedGuard

The RedGuard checks the received frame (which is “ReceivedInRED == TRUE” due to the DEMUX) against Table 472, Table 473 or Table 474. The used Table depends on the locally supported functionality.

Table 472 – Truth table for the RedGuard with full check

Input					Output		
Conditions of the receive port					Conditions of the transmit port		Local
FrameID [a]	Receive port	Phase	Data length	FrameSend Offset [c]	List of TX ports	Remote RTC3PSM	RED relay
FALSE	—	—	—	—	—	—	Discard [b]
TRUE	FALSE	—	—	—	—	—	Discard [b]
TRUE	TRUE	FALSE	—	—	—	—	Discard [b]
TRUE	TRUE	TRUE	FALSE	—	—	—	Discard [b]
TRUE	TRUE	TRUE	TRUE	FALSE	—	—	Discard [b]
TRUE	TRUE	TRUE	TRUE	TRUE	—	! RUN	Discard [b]
TRUE	TRUE	TRUE	TRUE	TRUE	Yes	RUN	Forward

[a] The FrameID shall be checked against the PDIRFrameData and PDIRBeginEndData.

[b] The bridge should discard the frame.

[c] The forwarding of the frame can be done keeping its given FrameSendOffset (TRUE) or not (FALSE).

Table 473 – Truth table for the RedGuard with reduced check

Input		Output		
Conditions of the receive port		Conditions of the transmit port		Local
FrameID [a]	FrameSend Offset [c]	List of TX ports	Remote RTC3PSM	RED relay
FALSE	—	—	—	Discard [b]
TRUE	FALSE	—	—	Discard [b]
TRUE	TRUE	—	! RUN	Discard [b]
TRUE	TRUE	Yes	RUN	Forward

[a] The FrameID shall be checked against PDIRFrameData and PDIRBeginEndData.

[b] The bridge should discard the frame.

[c] The forwarding of the frame can be done keeping its given FrameSendOffset (TRUE) or not (FALSE).

Table 474 – Truth table for the RedGuard with minimal check

Input	Output		
Conditions of the receive port	Conditions of the transmit port		Local
FrameID [a]	List of TX ports	Remote RTC3PSM	RED relay
FALSE	—	—	Discard [b]
TRUE	—	! RUN	Discard [b]
TRUE	Yes	RUN	Forward

[a] The FrameID shall be checked against PDIRFrameData (for local received and not forwarded frames) and PDIRBeginEndData.

[b] The bridge should discard the frame.

4.12.11.5 DFP Forwarding Protocol Machine

4.12.11.5.1 General

The DFP_RELAY arranges frames with the attribute DFP. A DFP frame is a special form of an RT_CLASS_3 frame.

DFP offers two different principles, inbound and outbound, for the handling of this kind of frames. Figure 154, Figure 155 and Figure 156 show the integration of DFP.

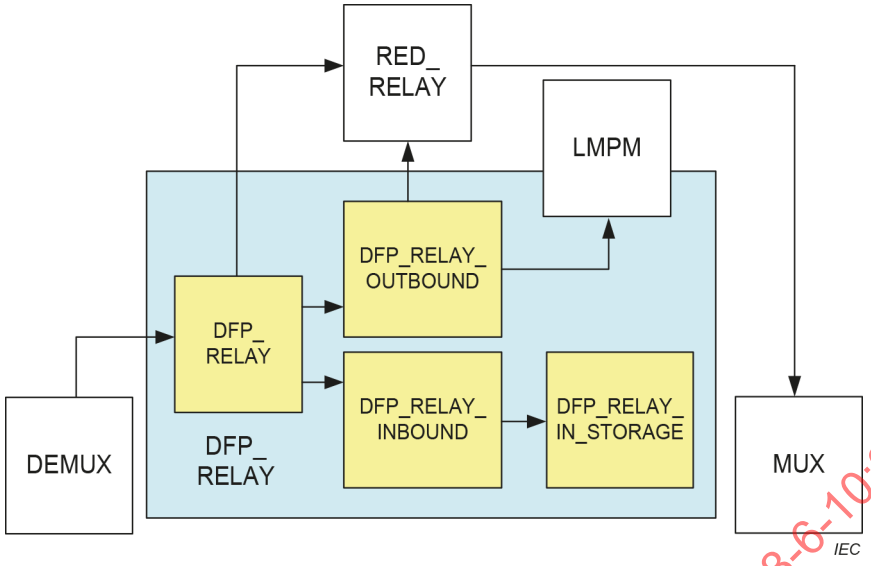


Figure 154 – Scheme of the DFP_RELAY

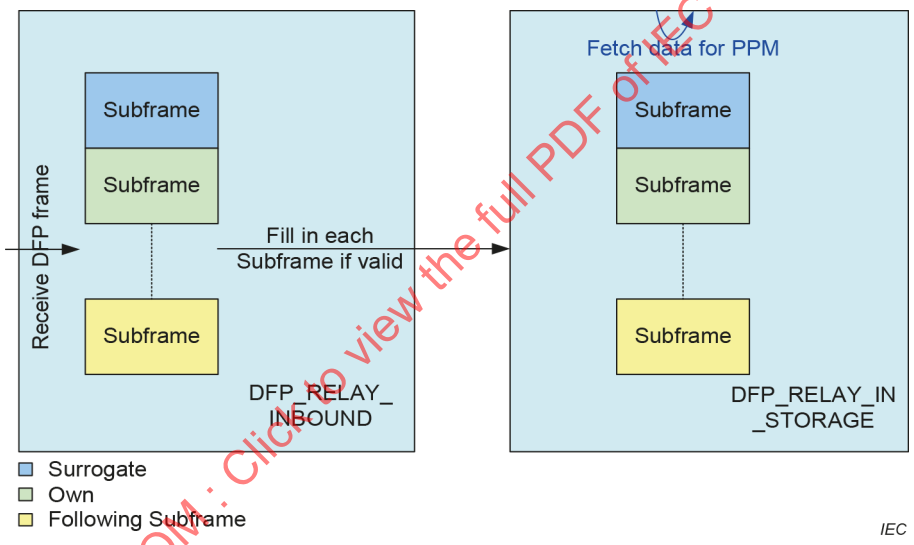


Figure 155 – Scheme of the DFP_RELAY_INBOUND and DFP_RELAY_IN_STORAGE

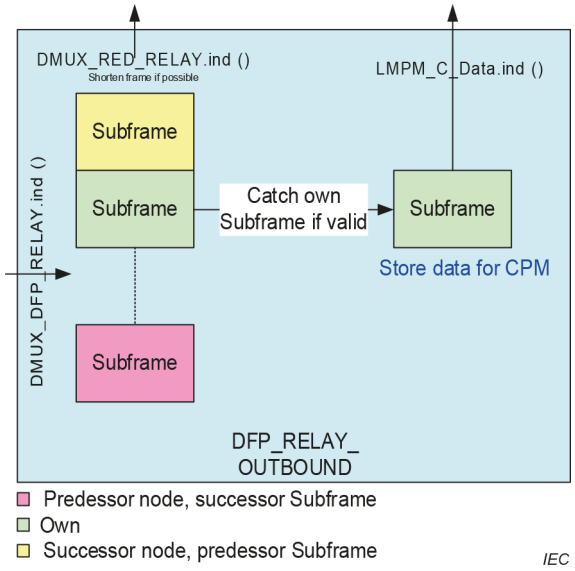


Figure 156 – Scheme of the DFP_RELAY_OUTBOUND

4.12.11.5.2 Startup

The DFP_RELAY shall, like the RED_RELAY, start with the persistently stored data from ealier AR establishment, after power on.

Thus, dynamic frame packing works independently of the IO controller startup sequence as soon as the RED_RELAY is active.

4.12.11.5.3 DFP Protocol Machine

4.12.11.5.3.1 Primitive definitions

4.12.11.5.3.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DFP Protocol Machine (DFP_RELAY) are described in the service definition and shown in Table 475.

Table 475 – Remote primitives issued or received by DFP_RELAY

Primitive	Source	Destination	Associated parameters	Functions
DMUX_DFP_RELAY.ind	DEMUX	DFP_RELAY	Port, Tstamp, DA, SA, C_SDU, ReceivedInRED	—
DFP_INBOUND.ind	DFP_RELAY	DFP_RELAY_INBOUND	Port, Tstamp, DA, SA, C_SDU, ReceivedInRED	—
DFP_OUTBOUND.ind	DFP_RELAY	DFP_RELAY_OUTBOUND	Port, Tstamp, DA, SA, C_SDU, ReceivedInRED	—

Primitive	Source	Destination	Associated parameters	Functions
DMUX_RED_RELAY.ind	DFP_RELAY	RED_RELAY	Port, Tstamp, DA, SA, C_SDU, ReceivedInRED	—

4.12.11.5.3.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DFP_RELAY are described in the service definition and shown in Table 476.

Table 476 – Local primitives issued or received by DFP_RELAY

Primitive	Source	Destination	Associated parameters	Functions
SyncStateChange_ind	PTCP	DFP_RELAY	state {SYNC, ASYNC}	—

4.12.11.5.3.2 State transition diagram

The state transition diagram of the DFP_RELAY is shown in Figure 157.

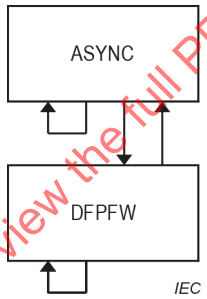


Figure 157 – State transition diagram of DFP_RELAY

States of the DFP_RELAY are:

ASYNC The node shall be synchronized, either locally as a sync master or remotely as a sync slave for the DFP forwarding which is based on the RED forwarding.

DFPFW DFP forwarding active

4.12.11.5.3.3 State machine description

This state machine receives the possible DFP frames from the DEMUX and forwards them to the assigned state machine.

Unknown RT_CLASS_3 frames, for example frames not defined by PDIRSubframeData, are forwarded to the RED_RELAY and not handled by the DFP_RELAY.

4.12.11.5.3.4 DFP_RELAY state table

The DFP_RELAY checks the received frame (which is “ReceivedInRED == TRUE” due to the DEMUX) against the state table shown in Table 477.

Table 477 – DFP_RELAY state table

#	Current State	Event /Condition =>Action	Next State
1	ASYNC	**SyncStateChange_ind ()** /SYNC => ignore	DFPFW
2	ASYNC	**SyncStateChange_ind ()** /!SYNC => ignore	ASYNC
3	DFPFW	**SyncStateChange_ind ()** /SYNC => ignore	DFPFW
4	DFPFW	**SyncStateChange_ind ()** /!SYNC => ignore	ASYNC
5	DFPFW	**DMUX_DFP_RELAY.ind ()** /DFPGuard () == OUTBOUND => DFP_OUTBOUND.ind ()	DFPFW
6	DFPFW	**DMUX_DFP_RELAY.ind ()** /DFPGuard () == INBOUND => DFP_INBOUND.ind ()	DFPFW
7	DFPFW	**DMUX_DFP_RELAY.ind ()** /DFPGuard () == RED_RELAY => DMUX_RED_RELAY.ind ()	DFPFW

4.12.11.5.3.5 Functions, Macros, Timers and Variables

Table 478 contains the functions, macros, timers, and variables used by the DFP_RELAY and their arguments and their descriptions.

Table 478 – Functions, Macros, Timers and Variables used by the DFP_RELAY

Name	Type	Function/Meaning
DFPGuard	Function	This function checks the received frame against the local information.

4.12.11.5.3.6 DFPGuard

The DFPGuard checks the received frame against Table 479.

Table 479 – Truth table for the DFPGuard

Input			Output			Comments
Local			Local			
FrameID [a]	Outbound	Inbound	RED_RELAY	OUTBOUND	INBOUND	
FALSE	—	—	TRUE	—	—	The FrameID is checked against the local DFP database. If it is not part of the database, then forward the frame to the RED_RELAY
TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	The local DFP database returns the frame attribute "OUTBOUND" for the given FrameID
TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	The local DFP database returns the frame attribute "INBOUND" for the given FrameID
TRUE	FALSE [b]	FALSE [b]	—	—	—	Parameterization fault detected during startup
TRUE	TRUE [b]	TRUE [b]	—	—	—	Parameterization fault detected during startup

[a] Checked against the PDIRSubframeData.

[b] Avoided by local checks.

4.12.11.5.4 DFP Inbound Protocol Machine

4.12.11.5.4.1 Primitive definitions

4.12.11.5.4.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DFP Protocol Machine (DFP_RELAY_INBOUND) are described in the service definition and shown in Table 480.

Table 480 – Remote primitives issued or received by DFP_RELAY_INBOUND

Primitive	Source	Destination	Associated parameters	Functions
DFP_INBOUND.ind	DFP_RELAY	DFP_RELAY_INBOUND	Port, Tstamp, DA, SA, N_SDU, ReceivedInRED	—

4.12.11.5.4.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DFP_RELAY_INBOUND are described in the service definition and shown in Table 481.

Table 481 – Local primitives issued or received by DFP_RELAY_INBOUND

Primitive	Source	Destination	Associated parameters	Functions
DFP_Activate_req	CMSU	DFP_RELAY_INBOUND	CREP	—
DFP_Close_req	CMSU	DFP_RELAY_INBOUND	CREP	—
DFP_Activate_cnf (+)	DFP_RELAY_INBOUND	CMSU	CREP	—
DFP_Close_cnf (+)	DFP_RELAY_INBOUND	CMSU	CREP	—
StoreSubframe_ind	DFP_RELAY_INBOUND	DFP_RELAY_IN_STORAGE	CREP, Subframe	Trigger the DFP STORAGE for further handling of the sub frame.

4.12.11.5.4.2 State transition diagram

The state transition diagram of the DFP_RELAY_INBOUND is shown in Figure 158.

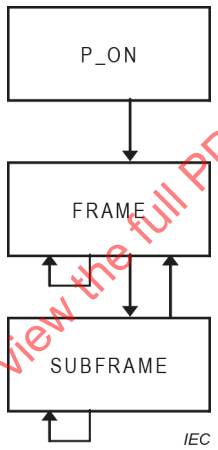


Figure 158 – State transition diagram of DFP_RELAY_INBOUND

States of the DFP_RELAY_INBOUND are:

P_ON	Data initialization
FRAME	Check DFP frame
SUBFRAME	Check DFP subframe

4.12.11.5.4.3 State machine description

This state machine receives the INBOUND DFP frames, identified by their FrameID, and checks them against the protocol definition supporting the two modes DistributedWatchDog active and passive.

For MRPD, two DFP layouts exist, identified by the FrameID.

4.12.11.5.4.4 DFP_RELAY_INBOUND state table

The DFP_RELAY_INBOUND shall ensure that the reception of a DFP frame is done in a correct way. The state table is shown in Table 482.

Table 482 – DFP_RELAY_INBOUND state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	**DFP_Activate_req ()** => DFP_Activate_cnf (+)	FRAME
2	FRAME	**DFP_INBOUND.ind ()** /InboundGuard () == FrameStructureValid => DFP_Check_Subframe ()	SUBFRAME
3	FRAME	**DFP_INBOUND.ind ()** /InboundGuard () == FrameStructureInvalid => ignore	FRAME
4	SUBFRAME	**DFP_Check_Subframe()** /InboundGuard () == SubframeStructureValid, SubframeDataValid => StoreSubframe_ind () DFP_Check_Subframe ()	SUBFRAME
5	SUBFRAME	**DFP_Check_Subframe()** /InboundGuard () == EndOfSubframe => ignore	FRAME
6	SUBFRAME	**DFP_Check_Subframe()** /InboundGuard () == SubframeStructureValid, SubframeDataInvalid => DFP_Check_Subframe ()	SUBFRAME
7	SUBFRAME	**DFP_Check_Subframe()** /InboundGuard () == SubframeStructureInvalid => ignore	FRAME
8	ANY	**DFP_Close_req ()** => DFP_Close_cnf (+)	P_ON

4.12.11.5.4.5 Functions, Macros, Timers and Variables

Table 483 contains the functions, macros, timers, and variables used by the DFP_RELAY_INBOUND and their arguments and their descriptions.

Table 483 – Functions, Macros, Timers and Variables used by the DFP_RELAY_INBOUND

Name	Type	Function/Meaning
DFP_Check_Subframe	Function	Trigger for further frame pARsing. The subframes are checked according to their sequence in the frame; each call of the function checks a particular subframe.
InboundGuard	Function	This function checks the received frame against the local information.
EndOfSubframe	Macro	The SFEndDelimiter is reached or no more data to be checked.
FrameStructureInvalid	Macro	The InboundGuard frame check returns Invalid.
FrameStructureValid	Macro	The InboundGuard frame check returns Valid.
SubframeDataInvalid	Macro	The InboundGuard subframe data check returns Invalid.
SubframeDataValid	Macro	The InboundGuard subframe data check returns Valid.
SubframeStructureInvalid	Macro	The InboundGuard subframe check returns Invalid.
SubframeStructureValid	Macro	The InboundGuard subframe check returns Valid.

4.12.11.5.4.6 InboundGuard

The InboundGuard checks the received frame against Table 484, Table 485, Table 486 and Table 487.

Table 484 – Truth table for the InboundGuard – frame check

Input	Output
Header Subframe. SFCRC16	**Frame check**
Invalid	Invalid
Valid	Valid

Table 485 – Truth table for the InboundGuard – subframe check

Input			Output
Subframe. Position [a]	**Subframe. Length** [b]	**Subframe. SFCRC16** [c]	**Subframe check**
Not found	—	—	Invalid
Found	FALSE	—	Invalid
Found	TRUE	Invalid	Invalid
Found	TRUE	Valid	Valid

[a] Checks whether the next expected subframe is received

[b] Checks whether the subframe has the expected length

[c] Checks whether the SFCRC16 is valid

Table 486 – Truth table for the InboundGuard – subframe data check

Input				Output
RestartFor-DistributedWDTime [a]	**Distributed-WatchDog-Factor** [b]	**Subframe. DataStatus. Valid** [c]	**Subframe. SFCycleCounter** [d]	**Subframe data check**
OFF	OFF	—	—	Valid
Running	—	—	—	Invalid
Expired / OFF	ON	Invalid	—	Invalid
Expired / OFF	ON	Valid	Old	Invalid
Expired / OFF	ON	Valid	New	Valid

[a] A running RestartForDistríbutedWDTime stops the storage of the subframe.

[b] The check of the subframe data is done by the endpoint if the distributed watchdog is disabled.

[c] The data is only stored if it is valid.

[d] Only newer subframes are stored.

Table 487 – Truth table for the InboundGuard – full check

Input			Output		
Frame check	Sub frame check	Sub frame data check	Frame-StructureValid	Subframe-StructureValid	Subframe-DataValid
Invalid	—	—	Invalid	—	—
Valid	Invalid	—	Valid	Invalid	—
Valid	Valid	Invalid	Valid	Valid	Invalid
Valid	Valid	Valid	Valid	Valid	Valid

4.12.11.5.5 DFP Inbound Storage Protocol Machine

4.12.11.5.5.1 General

From the consistency point of view there exists no difference between the rules for a frame or a subframe. The conveyance system shall insure that a frame is transported with frame consistency and a subframe is transported with subframe consistency.

4.12.11.5.5.2 Primitive definitions

4.12.11.5.5.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DFP Inbound Storage Protocol Machine (DFP_RELAY_IN_STORAGE) are described in the service definition and shown in Table 488.

Table 488 – Remote primitives issued or received by DFP_RELAY_IN_STORAGE

Primitive	Source	Destination	Associated parameters	Functions
AlarmNotification.ind	DFP_RELAY_IN_STORAGE	FAL	AREP, API, AlarmPriority, AlarmType, SlotNumber, SubslotNumber, AlarmSpecifier, ModuleIdentNumber, SubmoduleIdentNumber, AlarmItem	Inform the application that a concatenation fault has occurred.

4.12.11.5.5.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DFP_RELAY_IN_STORAGE are described in the service definition and shown in Table 489.

Table 489 – Local primitives issued or received by DFP_RELAY_IN_STORAGE

Primitive	Source	Destination	Associated parameters	Functions
DFP_Activate_req	CMSU	DFP_RELAY_IN_STORAGE	CREP	—
DFP_Close_req	CMSU	DFP_RELAY_IN_STORAGE	CREP	—
DFP_Activate_cnf(+)	DFP_RELAY_IN_STORAGE	CMSU	CREP	—
DFP_Close_cnf	DFP_RELAY_IN_STORAGE	CMSU	CREP	—
DFP_Error_ind	DFP_RELAY_IN_STORAGE	CMSU CTLSU	CREP, ERRCLS, ERRCODE	—
StoreSubframe_ind	DFP_RELAY_INBOUND	DFP_RELAY_IN_STORAGE	CREP, Subframe	Stores a received subframe.

4.12.11.5.5.3 State transition diagram

The state transition diagram of the DFP_RELAY_IN_STORAGE is shown in Figure 159.

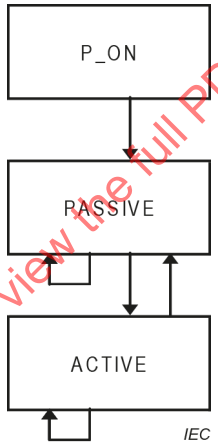


Figure 159 – State transition diagram of DFP_RELAY_IN_STORAGE

States of the DFP_RELAY_IN_STORAGE are:

P_ON Data initialization

PASSIVE Wait for first receive or for the timeout of the RestartForDistributedWDTime.

ACTIVE Check receive of a DFP subframe with distributed watchdog if activated. Additionally, check the concatenation of the subframes against the associated PPM time constraints and generate indications if necessary.

4.12.11.5.5.4 DFP_RELAY_IN_STORAGE state machine description

This state machine receives subframes and stores them for further use. The concatenation of the subframes to the locally generated frame is done in parallel to the storing of the data.

If a fault is detected, then the transmission of the frame should be aborted according to the rules of IEEE Std 802.3. In this case the last stored data of the subframe is still valid and usable.

4.12.11.5.5.5 DFP_RELAY_IN_STORAGE state table

The DFP_RELAY_IN_STORAGE shall control the reception of DFP frames. The corresponding state table is shown in Table 490.

Table 490 – DFP_RELAY_IN_STORAGE state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	**DFP_Activate_req ()** => Create list of sub frames in surrogate state List of successors If activated List of distributed watchdogs DFP_Activate_cnf(+)	PASSIVE
2	PASSIVE	**StoreSubframe_ind ()** /RestartForDistributedWDTime (Position) != Running => Update content of the stored sub frame (Position) If activated StartTimer (Distributed WDT (Position))	ACTIVE
3	PASSIVE	**StoreSubframe_ind ()** /RestartForDistributedWDTime (Position) == Running => ignore	PASSIVE
4	PASSIVE	**TimerExpired (RestartForDistributedWDTime (Position))** => ignore	PASSIVE
5	PASSIVE	**TimerExpired (Distributed WDT (Position))** => ignore	PASSIVE
6	PASSIVE	**TimerExpired (LateErrorDelayTimer)** => ignore	PASSIVE
7	ACTIVE	**StoreSubframe_ind ()** => Update content of the stored sub frame (Position) If activated StartTimer (Distributed WDT (Position))	ACTIVE
8	ACTIVE	**TimerExpired (Distributed WDT (Position))** => Set stored sub frame in surrogate state (Position) StartTimer (RestartForDistributedWDTime (Position))	PASSIVE
9	ACTIVE	**TimerExpired (RestartForDistributedWDTime (Position))** => ignore	ACTIVE
10	ACTIVE	**TickEvent ()** /Phase and FrameSendOffset of the associated PPM && Sub frames updated in this Phase => ignore	ACTIVE
11	ACTIVE	**TickEvent ()** /Phase and FrameSendOffset of the associated PPM && !Sub frames updated in this Phase && LateErrorDelayTimer == Running => StartTimer (LateErrorDelayTimer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
12	ACTIVE	**TickEvent ()** /Phase and FrameSendOffset of the associated PPM && !Sub frames updated in this Phase && LateErrorDelayTimer != Running => StartTimer (LateErrorDelayTimer) Alarm Priority := ALARM_HIGH or ALARM_LOW Alarm Type:= Dynamic Frame Packing problem notification User Structure Identifier := 0x8002 ChannelNumber := 0x8000 ChannelProperties.Specifier := Appears ChannelErrorType := “Dynamic frame packing function mismatch” ExtChannelErrorType := “Frame late error” AlarmNotification.ind ()	ACTIVE
13	ACTIVE	**TimerExpired (LateErrorDelayTimer)** => Alarm Priority := ALARM_HIGH or ALARM_LOW Alarm Type:= Dynamic Frame Packing problem notification User Structure Identifier := 0x8002 ChannelNumber := 0x8000 ChannelProperties.Specifier := Disappears ChannelErrorType := “Dynamic frame packing function mismatch” ExtChannelErrorType := “Frame late error” AlarmNotification.ind () //NOTE The value “Disappears” is used to illustrate the principle. The rules for ChannelProperties.Specifier apply.	ACTIVE
14	ANY	**DFP_Close_req ()** => Remove list of sub frames Remove stored diagnosis from Diagnosis ASE DFP_Close_cnf ()	P_ON

4.12.11.5.5.6 Functions, Macros, Timers and Variables

Table 491 contains the functions, macros, timers, and variables used by the DFP_RELAY_IN_STORAGE and their arguments and their descriptions.

Table 491 – Functions, Macros, Timers and Variables used by the DFP_RELAY_IN_STORAGE

Name	Type	Function/Meaning
Set stored sub frame in surrogate state	Function	This local function shall set the C_SDU of the subframe to zero and the DataStatus.Ignore of the subframe to TRUE; the other flags of the DataStatus and the SFCycleCounter should be zero.
StartTimer	Function	This local function is used to start or restart a timer.
TickEvent	Function	Locally generated event from a timer. The timer shall be controlled by the PTCP master or slave in case of synchronization or free running without synchronization. Parameters: CycleCounter, CycleOffset
TimerExpired	Function	This local function is issued if a timer expires.
Update content of the stored sub frame	Function	This local function updates the C_SDU, DataStatus and the SFCycleCounter of the subframe
Phase and FrameSendOffset of the associated PPM	Macro	This local macro uses the timing information of the associated PPM for a check against the information from TickEvent.
Sub frames updated in this Phase	Macro	This local macro checks whether the subframe was received in time to be concatenated to the local injected frame.

Name	Type	Function/Meaning
Distributed Watchdog Time	Timer	This timer is used to control the correct reception of valid data if activated.
LateErrorDelayTimer	Timer	This timer is used to limit the amount of reported “late errors” to a needed minimum. Time value: 1 s to 8 s Time base: 1 s
RestartForDistributedWDTime	Timer	This timer is used to protect a surrogate subframe for the detection of a fault by the IO controller. It is only active if the distributed watchdog is requested by the engineering.

4.12.11.5.6 DFP Outbound Protocol Machine

4.12.11.5.6.1 Primitive definitions

4.12.11.5.6.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DFP Outbound Protocol Machine (DFP_RELAY_OUTBOUND) are described in the service definition and shown in Table 492.

Table 492 – Remote primitives issued or received by DFP_RELAY_OUTBOUND

Primitive	Source	Destination	Associated parameters	Functions
DFP_OUTBOUND.ind	DFP_RELAY	DFP_RELAY_OUTBOUND	Port, Tstamp, DA, SA, C_SDU, ReceivedInRED	—
LMPM_C_Data.ind	DFP_RELAY_OUTBOUND	—	CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status, ReceivedInRED	Convey the own subframe to the local interface for further handling in the associated CPM.
DMUX_RED_RELAY.ind	DFP_RELAY_OUTBOUND	RED_RELAY	Port, Tstamp, DA, SA, N_SDU, ReceivedInRED	—

4.12.11.5.6.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DFP_RELAY_OUTBOUND are described in the service definition and shown in Table 493.

Table 493 – Local primitives issued or received by DFP_RELAY_OUTBOUND

Primitive	Source	Destination	Associated parameters	Functions
DFP_Activate_cnf (+)	DFP_RELAY_OUTBOUND	CMSU	CREP	—
DFP_Close_cnf (+)	DFP_RELAY_OUTBOUND	CMSU	CREP	—
DFP_Activate_req	CMSU	DFP_RELAY_OUTBOUND	CREP	—
DFP_Close_req	CMSU	DFP_RELAY_OUTBOUND	CREP	—

4.12.11.5.6.2 State transition diagram

The state transition diagram of the DFP_RELAY_OUTBOUND is shown in Figure 160.

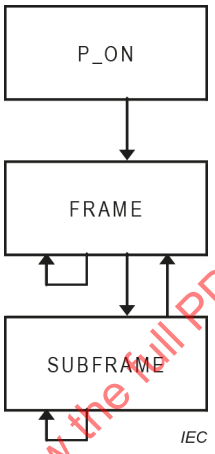


Figure 160 – State transition diagram of DFP_RELAY_OUTBOUND

States of the DFP_RELAY_OUTBOUND are:

P_ON	Data initialization
FRAME	Check the frame whether it has a valid structure and shall be shortened.
SUBFRAME	Search for the own subframe and shorten the frame if the structure is valid.

4.12.11.5.6.3 DFP_RELAY_OUTBOUND state machine description

This state machine receives a frame identified by the FrameID, checks whether the shortening is possible. If possible, the frame will be shortened before the own subframe and forwarded. The own subframe will be conveyed to the CPM. If not possible, the frame is forwarded to the RED_RELAY.

The DFP_RELAY_OUTBOUND shall ensure that the local portion of the DFP frames is handed over to the LMPM and the, where applicable, shortened frame is handed over to the RED_RELAY.

The frame should be shortened if SFIOCRProperties.SFCRC16 is set to 1 and the assigned subframe locally received after checking its SFCRC16.

The shortening shall be done by replacing the locally received subframe with a SFEndDelimiter, if needed Padding and an APDU_Status according to Table 494.

Table 494 – APDU_Status used if frame is shortened

Name	Function/Meaning
CycleCounter	Shall contain the “actual local value” derived from the PTCP synchronized SendClock.
TransferStatus	Shall be set to zero.
DataStatus	DataStatus.State:=1, DataStatus.Redundancy:=0, DataStatus.DataValid:=1, DataStatus.ProviderState:=1, DataStatus.StationProblemIndicator:=1, DataStatus.Ignore:=1

A locally assigned subframe shall be received

– after checking the FCS of the conveying frame if SFIOCRProperties.SFCRC16 is set to zero.
– after checking the SFCRC16 of the subframe if SFIOCRProperties.SFCRC16 is set to 1.

4.12.11.5.6.4 DFP_RELAY_OUTBOUND state table

The DFP_RELAY_OUTBOUND state table is shown in Table 495.

Table 495 – DFP_RELAY_OUTBOUND state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	**DFP_Activate_req ()** => DFP_Activate_cnf (+)	FRAME
2	FRAME	**DFP_OUTBOUND.ind ()** /OutboundGuard () == FrameStructureValid, FrameShortenImpossible => DMUX_RED_RELAY.ind ()	FRAME
3	FRAME	**DFP_OUTBOUND.ind ()** /OutboundGuard () == FrameStructureValid, FrameShortenPossible && FirstSubframeInFrame == OWN => DMUX_RED_RELAY.ind ()	FRAME
4	FRAME	**DFP_OUTBOUND.ind ()** /OutboundGuard () == FrameStructureValid, FrameShortenPossible && ! FirstSubframeInFrame == OWN => DFP_Check_Subframe ()	SUBFRAME
5	FRAME	**DFP_OUTBOUND.ind ()** /OutboundGuard () == FrameStructureInvalid => ignore //NOTE The frame shall be shortened after the last valid subframe by the MUX according to the rules of the IEEE Std 802.3	FRAME
6	SUBFRAME	**DFP_Check_Subframe ()** /OutboundGuard () == SubframeStructureValid && SFPosition.Position == OWN => N_SDU := ShortenFrame (N_SDU) LMPM_C_Data.ind () DMUX_RED_RELAY.ind ()	SUBFRAME

#	Current State	Event /Condition =>Action	Next State
7	SUBFRAME	**DFP_Check_Subframe ()** /OutboundGuard () == SubframeStructureValid && SFPosition.Position != OWN => DFP_Check_Subframe ()	SUBFRAME
8	SUBFRAME	**DFP_Check_Subframe ()** /OutboundGuard () == SubframeStructureInvalid => ignore //NOTE The frame shall be shortened after the last valid subframe by the MUX according to the rules of the IEEE Std 802.3	FRAME
9	ANY	**DFP_Close_req ()** => DFP_Close_cnf (+)	P_ON

4.12.11.5.6.5 Functions, Macros, Timers and Variables

Table 496 contains the functions, macros, timers, and variables used by the DFP_RELAY_OUTBOUND and their arguments and their descriptions.

Table 496 – Functions, Macros, Timers and Variables used by the DFP_RELAY_OUTBOUND

Name	Type	Function/Meaning
DFP_Check_Subframe	Function	Trigger for further frame pARsing. The subframes are checked according to their sequence in the frame; each call of the function checks a particular subframe.
OutboundGuard	Function	This local function checks the received frame against the local information.
ShortenFrame	Function	This local function shortens the N_SDU if the sequence of valid SFCRC16 is not broken.
FirstSubframeInFrame	Macro	This local macro returns the information of the first subframe in the frame. This information is needed to avoid shortening of a frame which contains only one subframe.
FrameStructureInvalid	Macro	The OutboundGuard frame check returns the value Invalid for the Frame structure.
SubframeStructureInvalid	Macro	The OutboundGuard subframe check returns the value Invalid for the Subframe structure.
OWN	Variable	This local variable contains the information about the own subframe.

4.12.11.5.6.6 OutboundGuard

The OutboundGuard checks the received frame against Table 497 and Table 498.

Table 497 – Truth table for the OutboundGuard – frame check

Input			Output	
SFIOCRProperties. SFCRC16 [a]	Header Subframe. SFCRC16 [b]	Structure before own Subframe [c]	Frame structure	Frame shorten
FALSE	—	—	Valid	Impossible
TRUE	Invalid	—	Invalid	—
TRUE	Valid	Invalid	Invalid	Impossible
TRUE	Valid	Valid	Valid	Possible

[a] The shortening of a frame is only possible if the SFCRC16 is valid. Thus, the subframes will only be checked if the SFCRC16 exists.

[b] The header SFCRC16 is checked before the subframe checking starts.

[c] The shortening is only possible if a consistent structure is found before the own subframe.

Table 498 – Truth table for the OutboundGuard – subframe check

Input				Output
Frame structure	Subframe. Position [a]	Subframe. Length [b]	Subframe. SFCRC16	Sub frame structure
Invalid	—	—	—	Invalid
Valid	Invalid	—	—	Invalid
Valid	Valid	Invalid	—	Invalid
Valid	Valid	Valid	Invalid	Invalid
Valid	Valid	Valid	Valid	Valid

[a] Check against the protocol definition, optionally against the structure from the PDIRSubframeData.

[b] Check against the protocol definition, optionally against the structure from the PDIRSubframeData.

4.12.11.6 Network Access Sender Protocol Machine

4.12.11.6.1 Primitive definitions

4.12.11.6.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Network Access Sender (MUX) are described in the service definition and shown in Table 499.

Table 499 – Remote primitives issued or received by MUX

Primitive	Source	Destination	Associated parameters	Functions
M_UNITDATA.req	MUX	MAC	frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, rif_information (optional), include_tag	—
M_UNITDATA.ind	MAC	MUX	time_stamp	—
LinkStateChange.ind	MAC	MUX	Port, Link_State	—
PortStateChange.ind	Virtual Bridge	MUX	Port, Port_State	—

4.12.11.6.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by MUX are described in the service definition and shown in Table 500.

Table 500 – Local primitives issued or received by MUX

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.12.11.6.2 State transition diagram

The state transition diagram of the MUX is shown in Figure 161.

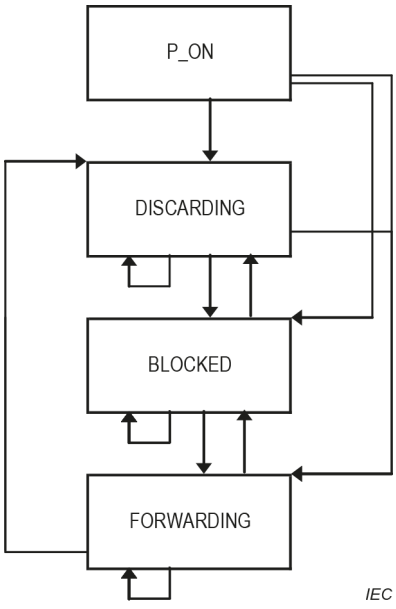


Figure 161 – State transition diagram of MUX

States of the MUX are:

P_ON	Wait for the startup Link and Port state and jump to the appropriate state
DISCARDING	All frames are discarded in this state
BLOCKED	Only network control or RT_CLASS_3 frames are forwarded in this state
FORWARDING	All frames are forwarded

4.12.11.6.3 State machine description

The MUX is responsible for mapping LMPM, RED_RELAY and MAC_RELAY (IEEE Std 802.1Q) services to MAC according to IEEE Std 802.3.

Additionally, the Queue handler as shown in Figure 169 is used to manage the priority queues.

4.12.11.6.4 MUX state table

The MUX state table is shown in Table 501.

Table 501 – MUX state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	/StateChecker () == FORWARDING => Period := GREEN	FORWARDING
2	P_ON	/StateChecker () == BLOCKED => Period := GREEN	BLOCKED
3	P_ON	/StateChecker () == DISCARDING => Period := GREEN	DISCARDING
4	P_ON	**M_UNITDATA.cnf ()** => ignore	P_ON
5	DISCARDING	**MUX_Schedule_req (CycleCounter, Tx_Period)** => ignore	DISCARDING
6	DISCARDING	**QueuesNotEmpty ()** => GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo, PRIO7, RED, PRIO6, …, PRIOR2-1) DISCARD_FRAME()	DISCARDING
7	DISCARDING	**M_UNITDATA.cnf ()** => ignore	DISCARDING
8	BLOCKED	**QueuesNotEmpty ()** /Period == GREEN && FrameSizeFits (NwC_Hi, NwC_Lo) => GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo) M_UNITDATA.req ()	BLOCKED
9	BLOCKED	**QueuesNotEmpty ()** /Period == GREEN && !FrameSizeFits (NwC_Hi, NwC_Lo) => ignore	BLOCKED

#	Current State	Event /Condition =>Action	Next State
10	BLOCKED	**M_UNITDATA.cnf ()** => ignore	BLOCKED
11	BLOCKED / FORWARDING	**MUX_Schedule_req (CycleCounter, Tx_Period)** => Period := Tx_Period	BLOCKED / FORWARDING
12	BLOCKED / FORWARDING	**QueuesNotEmpty ()** /Period == RED => GET_FRAME_FROM_QUEUE(RED) M_UNITDATA.req ()	BLOCKED / FORWARDING
13	FORWARDING	**QueuesNotEmpty ()** /Period == GREEN && FrameSizeFits (NwC_Hi, NwC_Lo, PRIO7, PRIO6, ..., PRIOR2-1) => GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo, PRIO7, PRIO6, ..., PRIOR2-1) M_UNITDATA.req ()	FORWARDING
14	FORWARDING	**QueuesNotEmpty ()** /Period == GREEN && !FrameSizeFits (NwC_Hi, NwC_Lo, PRIO7, PRIO6, ..., PRIOR2-1) => ignore	FORWARDING
15	FORWARDING	**M_UNITDATA.cnf ()** => ignore	FORWARDING
16	BLOCKED / FORWARDING	**QueuesNotEmpty ()** /Period != RED && FrameEgressFilter () => GET_FRAME_FROM_QUEUE() DISCARD_FRAME()	BLOCKED / FORWARDING
17	ANY	**LinkStateChange ()** /StateChecker () == DISCARDING => Period := GREEN	DISCARDING
18	ANY	**PortStateChange ()** /StateChecker () == DISCARDING => Period := GREEN	DISCARDING
19	ANY	**LinkStateChange ()** /StateChecker () == BLOCKED => ignore	BLOCKED
20	ANY	**PortStateChange ()** /StateChecker () == BLOCKED => ignore	BLOCKED
21	ANY	**LinkStateChange ()** /StateChecker () == FORWARDING => ignore	FORWARDING
22	ANY	**PortStateChange ()** /StateChecker () == FORWARDING => ignore	FORWARDING

4.12.11.6.5 Functions, Macros, Timers and Variables

Table 502 contains the functions, macros, timers, and variables used by the MUX and their arguments and their descriptions.

Table 502 – Functions, Macros, Timers and Variables used by MUX

Name	Type	Function/Meaning
FrameEgressFilter	Function	This local function checks whether the frame should be transmitted or dropped according to the IEEE Std 802.1Q egress filtering
FrameSizeFits	Function	This local function checks if the found frame fits into the remaining period till the next StartOfRED. It is only searched in the (…) queues from highest to lowest priority. The first non-fitting frame stops the search and FALSE is returned. The check includes the requirements of the corresponding truth table.
QueuesNotEmpty	Function	This local event is issued if an entry in any queue exists. An entry exists after a frame receive is detected and the CT or S&F switching has the transmit queue assigned.
StateChecker	Function	This local function checks the Link_State and Port_State for the possible next state and returns the result.
DISCARD_FRAME	Macro	Discards the selected frame
GET_FRAME_FROM_QUEUE	Macro	Delivers a frame out of (…) queues. This dequeuing always works from highest to lowest priority of the queues.
Period	Variable	This local variable stores the actual period for further use

4.12.11.6.6 Truth tables

4.12.11.6.6.1 FrameSizeFits rules truth table

Table 503 contains the truth table used by the MUX.

Please note that non-working combinations like “Distance greater than MaxFrameSize:=False”, “Well known protocols:= —“, and “Other protocols:=CT” are not shown in Table 503.

Table 503 – Truth table for FrameSizeFits

Input						Output
Period	**Distance greater than MaxFrameSize**	**Distance greater than YellowTime**	**Well known protocols**	**Other protocols**	**Frame size**	**FrameSizeFits**
GREEN	TRUE	TRUE	CT / S&F	CT / S&F	—	TRUE
GREEN	FALSE	TRUE	CT / S&F	—	—	TRUE
GREEN	FALSE	TRUE	—	S&F	Fits	TRUE
GREEN	FALSE	TRUE	—	S&F	Too big	FALSE
GREEN	FALSE	FALSE	S&F	S&F	Fits	TRUE
GREEN	FALSE	FALSE	S&F	S&F	Too big	FALSE
GREEN	FALSE	FALSE	CT	CT	—	FALSE
RED	—	—	—	—	—	—

4.12.11.6.6.2 StateChecker truth table

Table 504 contains the truth table used by the MUX.

Table 504 – Truth table for StateChecker

Input		Output
Link_State	**Port_State**	**StateChecker**
DOWN	DISCARDING	DISCARDING
DOWN	BLOCKED	DISCARDING
DOWN	FORWARDING	DISCARDING
UP	DISCARDING	DISCARDING
UP	BLOCKED	BLOCKED
UP	FORWARDING	FORWARDING

4.12.11.7 Network Access Receiver Protocol Machine

4.12.11.7.1 Primitive definitions

4.12.11.7.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Network Access Receiver (DEMUX) are described in the service definition and shown in Table 505.

Table 505 – Remote primitives issued or received by DEMUX

Primitive	Source	Destination	Associated parameters	Functions
M_UNITDATA.ind	MAC	DEMUX	Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier	—
DMUX_DEFRAG.ind	DEMUX	DEFRAG	Port, Tstamp, DA, SA, VLAN, N_SDU	—
DMUX_DFP_RELAY.ind	DEMUX	DFP_RELAY	Port, Tstamp, DA, SA, N_SDU, ReceivedInRED	—
DMUX_MAC_RELAY.ind	DEMUX	—	Port, Tstamp, DA, SA, N_SDU	—
DMUX_P_Data.ind	DEMUX	LMPM	Port, Tstamp, DA, SA, N_SDU	—
DMUX_RED_RELAY.ind	DEMUX	RED_RELAY	Port, Tstamp, DA, SA, N_SDU, ReceivedInRED	—

4.12.11.7.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by DEMUX are described in the service definition and shown in Table 506.

Table 506 – Local primitives issued or received by DEMUX

Primitive	Source	Destination	Associated parameters	Functions
DMUX_Schedule_req	SCHEDULER	DEMUX	Rx_Period	—
DMUX_Schedule_cnf	DEMUX	SCHEDULER	—	—

4.12.11.7.2 State transition diagram

The state transition diagram of the DEMUX is shown in Figure 162.

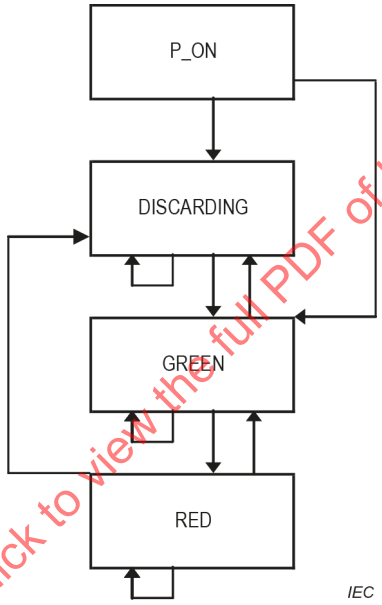


Figure 162 – State transition diagram of DEMUX

States of the DEMUX are:

P_ON	Power on
DISCARDING	Port state discarding
GREEN	Green period
RED	Red period

4.12.11.7.3 State machine description

The DEMUX is responsible for mapping MAC services according to IEEE Std 802.3 to LMPM, RED_RELAY and MAC_RELAY (IEEE Std 802.1Q).

The generation of the flag ReceivedInRED shall be done by checking the ReceiveTimeStamp made by the SHIM. If the value of the ReceiveTimeStamp is equal or greater than the RedOrangePeriodBegin and less or equal than the GreenPeriodBegin point in time, then the ReceivedInRED shall be set to TRUE, otherwise to FALSE.

4.12.11.7.4 DEMUX state table

The DEMUX state table is shown in Table 507.

Table 507 – DEMUX state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	/Port_State != Discarding => ignore	GREEN
2	P_ON	/Port_State == Discarding => ignore	DISCARDING
3	DISCARDING	**DMUX_Schedule_req ()** => DMUX_Schedule_cnf ()	DISCARDING
4	DISCARDING	**M_UNITDATA.ind ()** => ignore	DISCARDING
5	DISCARDING	**PortStateChange ()** /Port_State == Discarding => ignore	DISCARDING
6	DISCARDING	**PortStateChange ()** /Port_State != Discarding => ignore	GREEN
7	GREEN	**PortStateChange ()** /Port_State != Discarding => ignore	GREEN
8	GREEN	**PortStateChange ()** /Port_State == Discarding => ignore	DISCARDING
9	GREEN	**DMUX_Schedule_req ()** /Rx_Period == GREEN => DMUX_Schedule_cnf ()	GREEN
10	GREEN	**DMUX_Schedule_req ()** /Rx_Period == RED => DMUX_Schedule_cnf ()	RED
11	GREEN	**M_UNITDATA.ind ()** /FrameIngressFilter //NOTE Checking the ingress filter is always the first check! => ignore	GREEN
12	GREEN	**M_UNITDATA.ind ()** /FRAGMENT => DMUX_DEFRAG.ind ()	GREEN
13	GREEN	**M_UNITDATA.ind ()** /OTHER OR RED OR DFP => DMUX_MAC_RELAY.ind ()	GREEN

#	Current State	Event /Condition =>Action	Next State
14	GREEN	**M_UNITDATA.ind ()** /CTRL => Get_Receive_TimeStamp () DMUX_P_Data.ind ()	GREEN
15	RED	**PortStateChange ()** /Port_State != Discarding => ignore	RED
16	RED	**PortStateChange ()** /Port_State == Discarding => ignore	DISCARDING
17	RED	**DMUX_Schedule_req ()** /Rx_Period == GREEN => DMUX_Schedule_cnf ()	GREEN
18	RED	**DMUX_Schedule_req ()** /Rx_Period == RED => DMUX_Schedule_cnf ()	RED
19	RED	**M_UNITDATA.ind ()** /RED => ReceivedInRED := TRUE DMUX_RED_RELAY.ind ()	RED
20	RED	**M_UNITDATA.ind ()** /DFP => ReceivedInRED := TRUE DMUX_DFP_RELAY.ind ()	RED
21	RED	**M_UNITDATA.ind ()** /FRAGMENT => DMUX_DEFRAG.ind ()	RED
22	RED	**M_UNITDATA.ind ()** /OTHER => DMUX_MAC_RELAY.ind ()	RED
23	RED	**M_UNITDATA.ind ()** /CTRL => Get_Receive_TimeStamp () DMUX_P_Data.ind ()	RED

4.12.11.7.5 Functions, Macros, Timers and Variables

Table 508 contains the functions, macros, timers, and variables used by the DEMUX and their arguments and their descriptions.

Table 508 – Functions, Macros, Timers and Variables used by the DEMUX

Name	Type	Function/Meaning
Get_Receive_TimeStamp	Function	Get the receive timestamp from the Timestamp SHIM for this frame (see Figure 169)
CTRL	Macro	Network control/management, for example PTCP synchronization frames without “RT_CLASS_3 attribute”, LLDP, PTCP Line delay measurement, PTCP announce, media redundancy, …
DFP	Macro	RT_CLASS_3 frames with “DFP attribute” EtherType == 0x8892 FrameID == 0x0100…0x0FFF
FRAGMENT	Macro	Frame using the fragmentation protocol EtherType == 0x8892 FrameID == Fragmentation
FrameIngressFilter	Macro	Checks whether this frame is filtered by an ingress rule
OTHER	Macro	Any other frame
RED	Macro	RT_CLASS_3 frames EtherType == 0x8892 FrameID == 0x0100 …0x0FFF Special case: PTCP synchronization frames with “RT_CLASS_3 attribute” EtherType == 0x8892 FrameID == 0x0080
PortStateChange	Function	Conveys the info about the port state

4.13 IP suite

4.13.1 Overview

The utilization of the IETF RFC 768 (UDP), IETF RFC 791 (IP), IETF RFC 792 (ICMP), IETF RFC 826 (ARP), IETF RFC 2236 (IP Multicasting), IETF RFC 2474 (IP Extensions) is defined for this specification. Included are definitions for UDP ports and IP multicast addresses and utilization of IP header fields for RT_CLASS_UDP.

4.13.2 IP/UDP syntax description

4.13.2.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.13.2.2 IP/UDP APDU abstract syntax

Table 509 defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 509 shall represent the content of the DLSDU in Table 10.

Table 509 – IP/UDP APDU syntax

APDU name	APDU structure
IP-PDU	IPHeader [a], Data*
ICMP-PDU	IPHeader [a], ICMPHeader, Data*
UDP-PDU	IPHeader [b], UDPHeader, Data*
[a] The value of IPHeader.IP_Protocol shall be set to 1 to indicate ICMP. [b] The value of IPHeader.IP_Protocol shall be set to 17 to indicate UDP.	

Table 510 defines structures for substitutions of elements of the APDU structures shown in Table 509.

Table 510 – IP/UDP substitutions

Substitution name	Structure
IPHeader	IP_VersionIHL(0x45), IP_DifferentiatedServices, IP_TotalLength, IP_Identification, IP_Flags_FragOffset [a], IP_TTL, IP_Protocol, IP_HeaderChecksum, IP_SrcIPAddress, IP_DstIPAddress, [IP_Options] [b] The encoding of the fields shall be according to IETF RFC 791.
ICMPHeader	ICMP_Type, ICMP_Code, ICMP_HeaderChecksum
UDPHeader	UDP_SrcPort, UDP_DstPort, UDP_DataLength, UDP_Checksum [c]

[a] For UDP-RTC-PDU and UDP-RTA-PDU IP fragmentation shall not be used.

[b] The field IP_Options shall be omitted for UDP-RTC-PDU and UDP-RTA-PDU.

[c] The UDP_Checksum should be set to zero for RT_CLASS_UDP and RTA_CLASS_UDP to improve performance. It is not required for the receiver to check the UDP_Checksum for UDP-RTC-PDU and UDP-RTA-PDU.

4.13.3 IP/UDP transfer syntax

4.13.3.1 Coding section related to IP, ICMP and UDP

4.13.3.1.1 Coding section related to ICMP

4.13.3.1.1.1 Coding of the field ICMP_Type

The encoding of the fields shall be according to IETF RFC 792.

4.13.3.1.1.2 Coding of the field ICMP_Code

The encoding of the fields shall be according to IETF RFC 792.

4.13.3.1.1.3 Coding of the field ICMP_HeaderChecksum

The encoding of the fields shall be according to IETF RFC 792.

4.13.3.1.2 Coding section related to UDP

4.13.3.1.2.1 Coding of the field UDP_DataLength

The encoding of the fields shall be according to IETF RFC 768.

4.13.3.1.2.2 Coding of the field UDP_Checksum

The encoding of the fields shall be according to IETF RFC 768.

4.13.3.1.2.3 Coding of the field UDP_SrcPort

The encoding of the fields shall be according to IETF RFC 768. Defined values are shown in Table 511.

Table 511 – UDP_SrcPort

Value (hexadecimal)	Meaning	Usage
0x8892	IANA_PNIO_UDP_UNICAST_PORT	UDP-RTC-PDU and UDP-RTA-PDU
0x8893	IANA_PNIO_UDP_MULTICAST_PORT	Reserved
0x8894	IANA_PNIO_EPM_PORT	NDREPMapLookupReq or NDREPMapLookupFreeReq
0xC000 – 0xFFFF	IANA_FREE_PORT Is used as dynamic and/or private port	PROFINETIOServiceReqPDU and PROFINETIOServiceResPDU

4.13.3.1.2.4 Coding of the field UDP_DstPort

The encoding of the fields shall be according to IETF RFC 768. Defined values are shown in Table 512.

Table 512 – UDP_DstPort

Value (hexadecimal)	Meaning	Usage
0x8892	IANA_PNIO_UDP_UNICAST_PORT	UDP-RTC-PDU and UDP-RTA-PDU
0x8893	IANA_PNIO_UDP_MULTICAST_PORT	Reserved
0x8894	IANA_PNIO_EPM_PORT	NDREPMapLookupReq or NDREPMapLookupFreeReq, may be used for initial call from IO controller to IO device and vice versa.
0xC000 – 0xFFFF	IANA_FREE_PORT Is used as dynamic and/or private port	PROFINETIOServiceReqPDU and PROFINETIOServiceResPDU

4.13.3.1.3 Coding section related to IP

4.13.3.1.3.1 Coding of the field IP_DstIPAddress

The encoding of the fields shall be according to IETF RFC 791, IETF RFC 2236 and IETF RFC 2365. Defined values are shown in Table 513 and Table 514.

Table 513 – IP_DstIPAddress

IP address	Range	Meaning
224.0.0.34	subnet	SUBNET_MULTICAST_IP_ADDRESS_FOR_DISCOVERY
—	site	SITE_MULTICAST_IP_ADDRESS_FOR_DISCOVERY

Table 514 – IP Multicast DstIPAddress according to IETF RFC 2365

Multicast IP address	Associated Multicast MAC address	Associated FrameID	Usage
239.0.0.0 – 239.192.247.255	—	—	Not used
239.192.248.0	01-00-5E-40-F8-00	0xF800	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.192.248.1 – 239.192.251.254	01-00-5E-40-F8-01 – 01-00-5E-40-FB-FE	0xF801 – 0xFBFE	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.192.251.255	01-00-5E-40-FB-FF	0xFBFF	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.193.252.0 – 239.255.255.255	—	—	Not used

4.13.3.1.3.2 Coding of the field IP_VersionIHL

The encoding of the fields shall be according to IETF RFC 791.

4.13.3.1.3.3 Coding of the field IP_DifferentiatedServices

The encoding of the fields shall be according to IETF RFC 2474 and IETF RFC 2475.

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0 – 5: IP_DifferentiatedServices.DSCP
This field shall be coded with the values according to Table 515.

Table 515 – IP_DifferentiatedServices.DSCP

Value (hexadecimal)	Name	Meaning
0x0	CS0 / DEFAULT / BEST EFFORT	See IETF RFC 2474 and IETF RFC 2475 Default
0x8	CS1	See IETF RFC 2474 and IETF RFC 2475 May be for example used for IP traffic defined by this document (IP/UDP/RPC) to allow bridge resource protection by assigning traffic classes at the bridges.
0xA	AF11	
0xC	AF12	
0xE	AF13	
other	—	See IETF RFC 2474 and IETF RFC 2475

Bit 5 – 7: IP_DifferentiatedServices.ECN
This field shall be coded with the values according to Table 516.

Table 516 – IP_DifferentiatedServices.ECN

Value (hexadecimal)	Meaning
0x0	See IETF RFC 2474 and IETF RFC 2475 Default
0x1	See IETF RFC 2474 and IETF RFC 2475
0x2	
0x3	

4.13.3.1.3.4 Coding of the field IP_TotalLength

The encoding of the fields shall be according to IETF RFC 791.

4.13.3.1.3.5 Coding of the field IP_Identification

The encoding of the fields shall be according to IETF RFC 791.

4.13.3.1.3.6 Coding of the field IP_Flags_FragOffset

The encoding of the fields shall be according to IETF RFC 791.

4.13.3.1.3.7 Coding of the field IP_TTL

The encoding of the fields shall be according to IETF RFC 791.

4.13.3.1.3.8 Coding of the field IP_Protocol

The encoding of the fields shall be according to IETF RFC 791.

4.13.3.1.3.9 Coding of the field IP_HeaderChecksum

The encoding of the fields shall be according to IETF RFC 791.

4.13.3.1.3.10 Coding of the field IP_SrcIPAddress

The encoding of the fields shall be according to IETF RFC 791.

4.13.3.1.3.11 Coding of the field IP_Options

The encoding of the fields shall be according to IETF RFC 791.

4.13.4 ARP

4.13.4.1 General

The utilization of the IETF RFC 826 (ARP) and the control of the ARP cache should be optimized to reduce the switchover time between two nodes using the same IP address at different times.

Additionally, a static ARP cache entry for used IPAddresses reduces the seen ARP load on the network.

Both IO controller and IO device use ARP to convert an IPAddress into a MACAddress. As a proxy for the IO device, the IO controller uses a variant of “ARP probe” (see 5.6.4.12 (CTLDINA) and IETF RFC 5227) to check whether an IO device assigned IPAddress is already in use.

4.13.4.2 ARP Cache Control Machine

4.13.4.2.1 Primitive definitions

4.13.4.2.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by ARP Cache Control Machine (ACCM) are described in the service definition and shown in Table 517.

Table 517 – Remote primitives issued or received by ACCM

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.13.4.2.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by ACCM are described in the service definition and shown in Table 518.

Table 518 – Local primitives issued or received by ACCM

Primitive	Source	Destination	Associated parameters	Functions
ACCM_req	CTLDINA, CMCTL CMSU	ACCM	Command:={Add, Remove}, IPAddress, MACAddress	This service primitive requests the adding or removing of a static entry to/from the ARP cache.
ACCM_cnf	ACCM	CTLDINA, CMCTL, CMSU	Command:={Add, Remove}	This service primitive confirms that the adding or removing of a static entry to/from the ARP cache is done.

4.13.4.2.2 State transition diagram

The state transition diagram of the ACCM is shown in Figure 163.

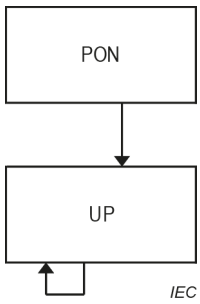


Figure 163 – State transition diagram of ACCM

States of the ACCM are:

PON	Init after power-on is done by the IP suite ASE.
UP	Wait for a request to add or remove a static entry to/from the ARP cache

4.13.4.2.3 State machine description

The ARP cache control machine defines the adding and removing of ARP cache entries to/from the static area of the ARP cache.

The static entries of the ARP cache are flushed with each Power-On.

The minimum number of static entries is equal or higher than either

– the number of supported IODs for an IOC, or
– the number of concurrent ARs for an IOD.

If both are supported concurrently, the sum of entries shall be supported.

More entries, static and/or dynamic, are needed for other protocols or additional communication partners.

4.13.4.2.4 ACCM state table

Table 519 contains the state table used by ACCM.

Table 519 – ACCM state table

#	Current State	Event /Condition =>Action	Next State
1	PON	=> InitARPcache	UP
2	UP	**ACCM_req ()** /Command == Add => AddStaticARPcacheEntry (IPAddress, MACAddress) ACCM_cnf ()	UP
3	UP	**ACCM_req ()** /Command == Remove => RemoveStaticARPcacheEntry (IPAddress) ACCM_cnf ()	UP

4.13.4.2.5 Functions, Macros, Timers and Variables

Table 520 contains the functions, macros, timers, and variables used by the ACCM and their arguments and their descriptions.

Table 520 – Functions, Macros, Timers and Variables used by the ACCM

Name	Type	Function/Meaning
AddStaticARPcacheEntry	Function	This local function adds a static entry to the ARP cache. A static entry invalidates any dynamic entry containing the IPAddress of the static entry.
InitARPcache	Function	This local function initializes the ARP cache
RemoveStaticARPcacheEntry	Function	This local function removes a static entry from the ARP cache.

4.14 Domain name system

4.14.1 General

The utilization of the IETF RFC 1034 (DNS) is defined for this specification. It includes the usage and the content syntax of the domain name.

4.14.2 Primitive definitions

4.14.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DNS are described in the service definition and shown in Table 521.

High level interfaces may offer a GetHostByName service instead of the QNAME service.

Table 521 – Remote primitives issued or received by DNS

Primitive name	Source	Destination	Associated parameters	Functions
DNS_QueryName.req	DNS	DNS client	QNAME := StationName.RealStationName or StationName.AliasStationName, QCLASS, QTYPE	Resolve name with IP address
DNS_QueryName.cnf	DNS server	DNS	QNAME := StationName.RealStationName or StationName.AliasStationName, QCLASS, QTYPE, Answer := IPAddress	Resolve name with IP address

4.14.2.2 Primitives exchanged between local machines

The local primitives including their associated parameters issued or received by DNS are described in the service definition and shown in Table 522.

Table 522 – Local primitives issued or received by DNS

Primitive name	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.14.3 DNS state transition diagram

See IETF RFC 1034 (DNS).

4.14.4 State machine description

See IETF RFC 1034 (DNS).

4.14.5 DNS state table

See IETF RFC 1034 (DNS).

4.14.6 Functions, Macros, Timers and Variables

Table 523 contains the functions, macros, timers, and variables used by the DNS and their arguments and their descriptions.

Table 523 – Functions, Macros, Timers and Variables used by the DNS

Name	Type	Function/Meaning
—	—	—

4.15 Dynamic host configuration

4.15.1 General

The utilization of the IETF RFC 2131 (DHCP) is defined for this specification. It includes the usage and the content syntax of Client Identifier.

NOTE A tight coupling between DNS and DHCP server prevents NameOfStation vs. IP address mismatch.

4.15.2 Primitive definitions

4.15.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by DHCP are described in the service definition and shown in Table 524.

Table 524 – Remote primitives issued or received by DHCP

Primitive name	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.15.2.2 Primitives exchanged between local machines

The local primitives including their associated parameters issued or received by DHCP are described in the service definition and shown in Table 525.

Table 525 – Local primitives issued or received by machines

Primitive	Source	Destination	Associated parameters	Functions
PN_DHCP_DISCOVER_req	DHCP	DHCP-ASE	Options	—
PN_DHCP_DISCOVER_cnf	DHCP-ASE	DHCP	—	—
PN_DHCP_OFFER_ind	DHCP-ASE	DHCP	IP_Para, ListOfData	—

4.15.3 DHCP state transition diagram

The state transition diagram of the DHCP is shown in Figure 164. The IETF RFC 2131 (DHCP) state machine is subordinated to state Running.

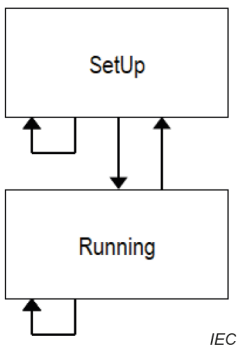


Figure 164 – State transition diagram of DHCP

States of the DHCP are:

SetUp	Wait until the subordinate state machine (IETF RFC 2131 – DHCP) shall be started.
Running	The subordinate DHCP state machine is running and should offer data for the CIM database. The state is switched back to SetUp if no more data from the subordinate DHCP server is needed.

4.15.4 State machine description

The DHCP state machine requests an IP-suite from the network, if activated.

The received dataset is stored in the CIM database.

4.15.5 DHCP state table

Table 526 contains the description of the DHCP state machine.

Table 526 – DHCP state table

#	Current State	Event /Condition =>Action	Next State
1	SetUp	**CM_DatabaseUpdate_ind** /LocalDHCPenable => Option := Start PN_DHCP_DISCOVER_req ()	Running
2	SetUp	**CM_DatabaseUpdate_ind** /! LocalDHCPenable => ignore	SetUp
3	SetUp	**PN_DHCP_DISCOVER_cnf ()** => ignore	SetUp
4	SetUp	**PN_DHCP_OFFER.ind ()** => ignore	SetUp
5	Running	**CM_DatabaseUpdate_ind** /LocalDHCPenable => ignore	Running
6	Running	**CM_DatabaseUpdate_ind** /! LocalDHCPenable => Option := Stop PN_DHCP_DISCOVER_req ()	SetUp
7	Running	**PN_DHCP_DISCOVER_cnf ()** => ignore	Running
8	Running	**PN_DHCP_OFFER.ind ()** /CheckAPDU () == InValid => ignore	Running
9	Running	**PN_DHCP_OFFER.ind ()** /CheckAPDU () == Valid => Store Data in DCP ASE	Running

#	Current State	Event /Condition =>Action	Next State
10	Running	**PN_DHCP_OFFER.ind ()** /CheckAPDU () == Valid, Name, IP || CheckAPDU () == Valid, Name || CheckAPDU () == Valid, IP => Store Data in DCP ASE CIM_DatabaseUpdate_ind	Running
11	Running	**PN_DHCP_OFFER.ind ()** /CheckAPDU () == Valid, Name, IP, RejectARactive || CheckAPDU () == Valid, IP, RejectARactive => Store Data in DCP ASE //NOTE IP and Name shall be adopted CIM_DatabaseUpdate_ind	Running

4.15.6 Functions, Macros, Timers and Variables

Table 527 contains the functions, macros, timers, and variables used by the DHCP and their arguments and their descriptions. The return values of macro CheckAPDU is shown in Table 528.

Table 527 – Functions, Macros, Timers and Variables used by the DHCP

Name	Type	Function/Meaning
CheckAPDU	Function	This local function checks the block structure and the parameters according to the coding and checking rules. This function returns Valid only if the whole PDU is valid.
LocalDHCPenable	Macro	This local macro requests the DHCP state from the DCP ASE and returns NIL if it does not exist. Return NIL if the Ipstack is not available.

Table 528 – Return values of macro CheckAPDU

Value	Meaning
InValid	InValid states that the frame is incorrect according to this document.
Valid	Valid states that the frame is correct according to this document. Valid allows additional return values to provide further details.
IP	IP states that the frame contains an IP-suite.
Name	Name states that the frame contains the NameOfStation.
RejectARactive	Shows that an AR exists and the AR should not aborted.

4.16 Simple network management

4.16.1 General

SNMPv1 supports only access to the 32 bit portion of statistic counters, which are historically defined as 32 bit values, independent of the device internal implementation.

SNMPv2 according to IETF RFC 1901 and IETF RFC 2578 supports access to 64 bit statistic counters which are suitable or even mandatory for higher link speeds as explained in IETF RFC 2863.

Thus, if 64 bit statistic counters are needed, the support of SNMPv2 limited to the services defined for SNMPv1 is needed as shown in Table 529.

Table 529 – SNMP service overview

Primitive name	SNMP version	Feature	Function/Meaning
Get	SNMPv1, SNMPv2	Mandatory	Manager requests some managed objects
GetNext	SNMPv1, SNMPv2	Mandatory	Manager requests the next managed object of the MIB
Response	SNMPv1, SNMPv2	Mandatory	Agent replies to the request of the manager
Set	SNMPv1, SNMPv2	Mandatory	Manager changes data on the device being managed
Trap	SNMPv1, SNMPv2	Optional	Agent alerts manager of special events (e.g. link down)
GetBulk	SNMPv2	Optional	Manager requests portions of a Table of a MIB
Notification	SNMPv2	Optional	Agent alerts manager of generic events
Inform	SNMPv2	Optional	Agent notification with manager response
Report	SNMPv2	Optional	Engine to engine communication

4.16.2 MIB overview

The utilization of the

– IETF RFC 1213 (MIB-II MIB),
– IETF RFC 2674 (Bridges with traffic classes),
– IETF RFC 2863 (IF-MIB),
– IETF RFC 3418 (SNMP),
– IETF RFC 3621 (Power over Ethernet MIB),
– IETF RFC 4836 (MAU-MIB), and
– IEEE Std 802.1AB (LLDP-MIB)

is defined for this specification.

Furthermore, the PNIO MIB is specified in 4.16.5 and LLDP EXT MIB is specified in 4.16.8.

4.16.3 MIB access

SNMP write requests (SNMP_SetRequest) to MIB objects should be rejected, except for the writeable OIDs shown in Table 530.

4.16.4 IETF RFC 1213-MIB

OIDs according to Table 530 shall be supported if SNMP is supported.

Table 530 – List of supported IETF RFC 1213-MIB objects

OID value	OID name	Accessible	Persistence
1.3.6.1.2.1.1.1	sysDescr [a]	Read	—
1.3.6.1.2.1.1.2	sysObjectID	Read	—
1.3.6.1.2.1.1.3	sysUpTime	Read	—
1.3.6.1.2.1.1.4	sysContact	Read / Write	YES
1.3.6.1.2.1.1.5	sysName	Read / Write	YES
1.3.6.1.2.1.1.6	sysLocation	Read / Write	YES
1.3.6.1.2.1.1.7	sysServices	Read	—

[a] Coding according "SystemIdentification" (preferred) or "ExtendedSystemIdentification" recommended.

4.16.5 Enterprise number for PNIO MIB

The coding of the field enterprise number shall be according to Table 531.

Table 531 – Enterprise number

Value (decimal)	Meaning
24 686	Enterprise number (OID root) for the PNIO MIB

4.16.6 MIB cross reference

The value of (LLDP-MIB.)"lldpLocPortDesc" for each port or interface shall be unique within one device to support cross reference to the IETF RFC 1213-MIB as shown in Table 532.

The link between the IETF RFC 1213-MIB and the LLDP-MIB shall be done according to Table 532.

Table 532 – Cross reference – MIBs

LLDP-MIB	Rule	IETF RFC 1213-MIB
lldpMIB.lldpObjects.lldpLocalSystemData.-lldpLocManAddrTable.lldpLocManAddrIfID	==	mib-2.interfaces.ifTable.ifIndex
lldpMIB.lldpObjects.lldpLocalSystemData.-lldpLocPortTable.lldpLocPortDesc	==	mib-2.interfaces.ifTable.ifIndex.ifDescr

The link between PDPortDataAdjust and the IETF RFC 1213-MIB shall be done according to Table 533.

Table 533 – Cross reference – PDPortDataAdjust

PDPortDataAdjust	Rule	IETF RFC 1213-MIB
LinkState.Link == DOWN	==	mib-2.interfaces.ifTable.ifIndex.ifAdminStatus

4.16.7 Behavior in case of modular built bridges

Modular built bridges (or devices) with pluggable ports or interfaces shall have either

– no entry, or
– an entry with ifOperStatus:=notPresent

for unplugged ports or interfaces stated in the ifTable of MIB-2.

4.16.8 LLDP EXT MIB

See Annex U.

4.17 Network configuration

4.17.1 Overview

Network Configuration is done by using NETCONF and YANG.

Figure 165 illustrates the structuring of a NME domain. One or more NME Domain Management Entities (NME), each a functional superset of an Centralized Network Configuration (CNC), together with one or more Query Stream entities integrated into IO controllers, each a functional superset of an Centralized User Configuration (CUC), are used to configure the network and to establish the streams.

The NME consist of:

- Centralized Network Configuration (CNC),
  - Resource Allocation Entity (RAE)
  - Path Entity (PE)
  - Topology Discovery Entity (TDE), including topology verification
  - Network Provisioning Entity (NPE)
  - Sync Tree Entity (STE)
- …

The NME uses the CNC (see Figure 165) to calculate and establish the path of a stream. Netconf is used for transport and YANG is used as database.

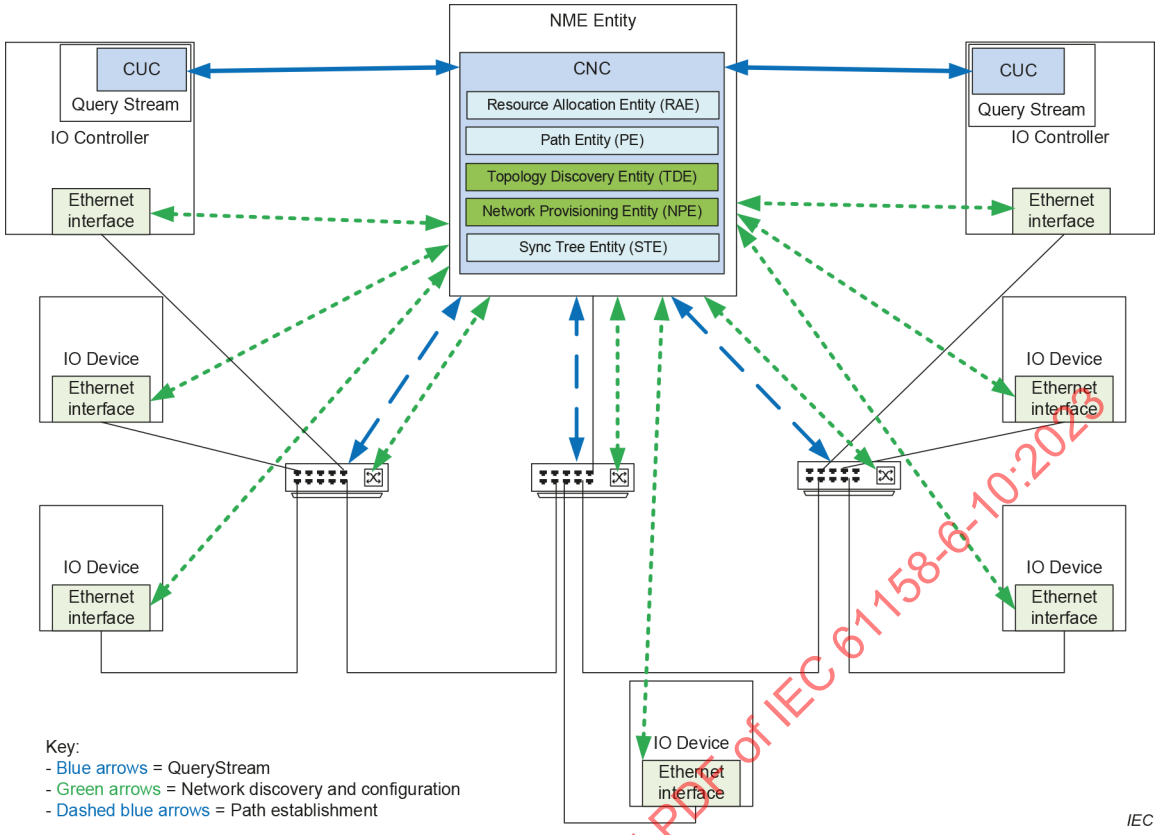


Figure 165 – Network Management Entity

4.17.2 NETCONF

The utilization of the

– IETF RFC 3535 (Informational: Background),
– IETF RFC 6244 (NETCONF+YANG Architectural Overview),
– IETF RFC 6241 (NETCONF protocol),
– IETF RFC 5539 (Transport mapping),
– IETF RFC 5277 (Notifications),
– IETF RFC 6243 (With defaults),
– IETF RFC 6470 (Base Notifications), and
– IETF RFC 6536 (NETCONF Access Control Model)
– IETF RFC 8342 (Network Management Datastore Architecture)
– ...

is defined for this specification.

Figure 166 shows the used Network Management Datastore Architecture for configuration. Not all shown datastores or views may be available.

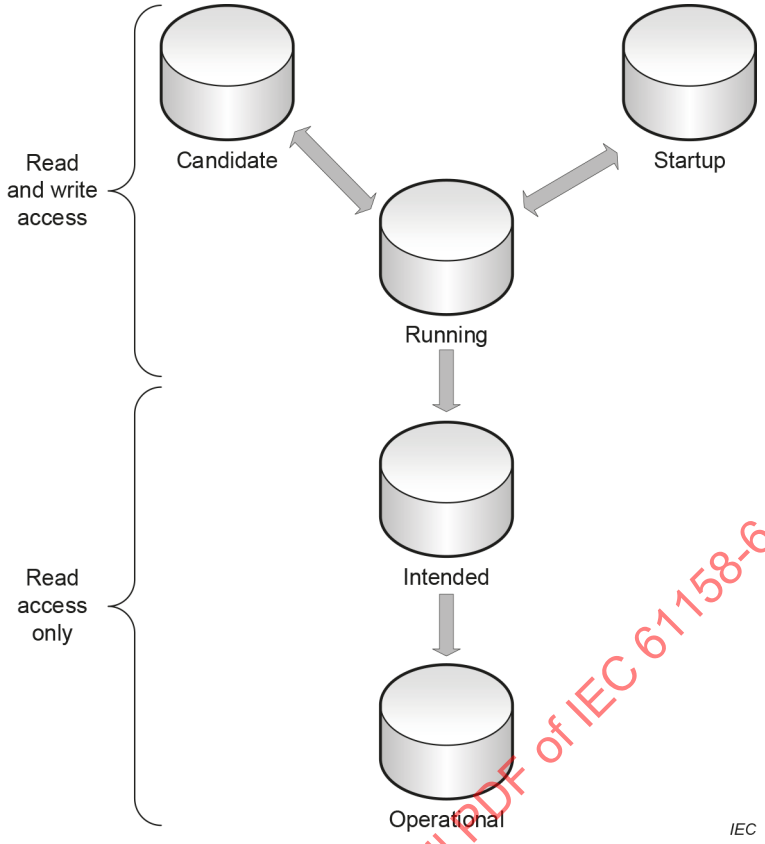


Figure 166 – NMDA model for network management

4.17.3 YANG

4.17.3.1 General

The utilization of the

– IETF RFC 6020 (YANG Base Specification),
– IETF RFC 6021 (YANG Types),
– IETF RFC 6087 (Guidelines for YANG Authors and Reviewers),
– IETF RFC 6110 (Mapping and Validating YANG),
– IETF RFC 6244 (NETCONF+YANG Architectural Overview), and
– …

is defined for this specification.

4.17.3.2 Bridge component

Figure 167 shows a selection of YANG models used to configure a bridge component and its host.

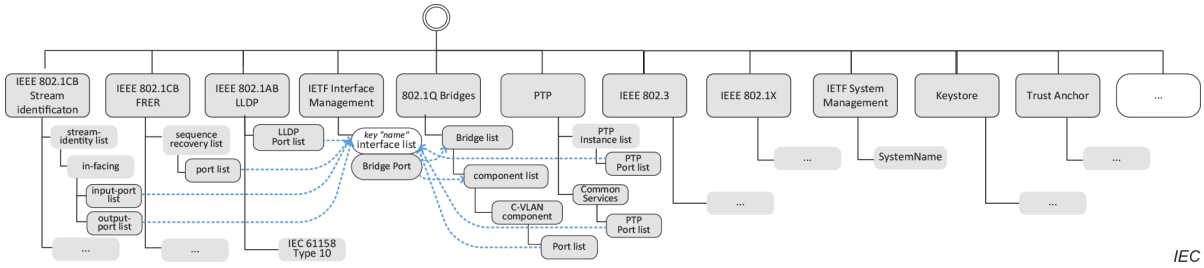


Figure 167 – YANG models of a bridge component

4.17.3.3 End station component

Figure 168 shows a selection of YANG models used to configure an end station component and its host.

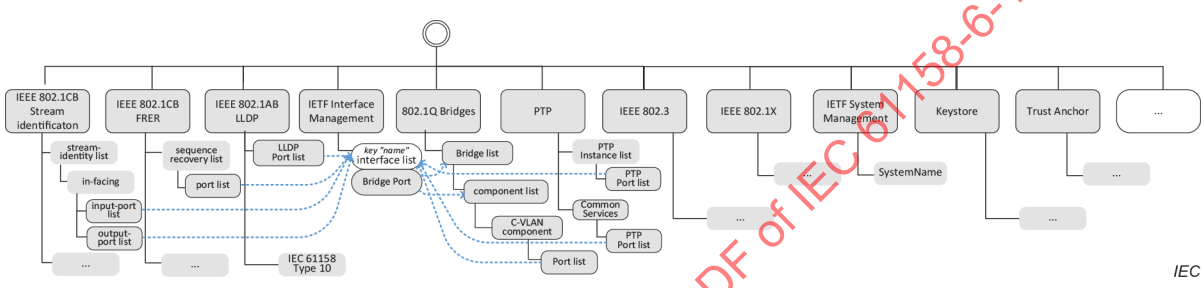


Figure 168 – YANG models of an end station component

4.18 Common DLL Mapping Protocol Machines

4.18.1 Overview

The DLL Mapping Protocol Machine (DMPM) consists of several protocol machines:

- End station component according to 4.12
  - Handling of the services invoked by other Application Layer entities (LMPM)
- Bridge component according to 4.12
  - Forwarding for RT_CLASS_3 (RED_Relay)
  - Forwarding for DFP (DFP_Relay)

Forwarding of standard frames shall be according to IEEE Std 802.1Q.

- Mapping to Media Access Control interface (MAC) and send/receive functions for synchronization messages

Invoking of MAC shall be according to IEEE Std 802.3.

- Fragmentation and reassembling (FRAG and DEFRAG)
- SYNC_SHIM to add precise time stamps for synchronization

The Interface between the service handler (LMPM), forwarding machines and mapping to Media Access Control (MAC) consists of a set of queues and global data.

Figure 169 illustrates the structuring of the protocol machines for the DMPM for a device and Annex H describes the upper protocol layers.

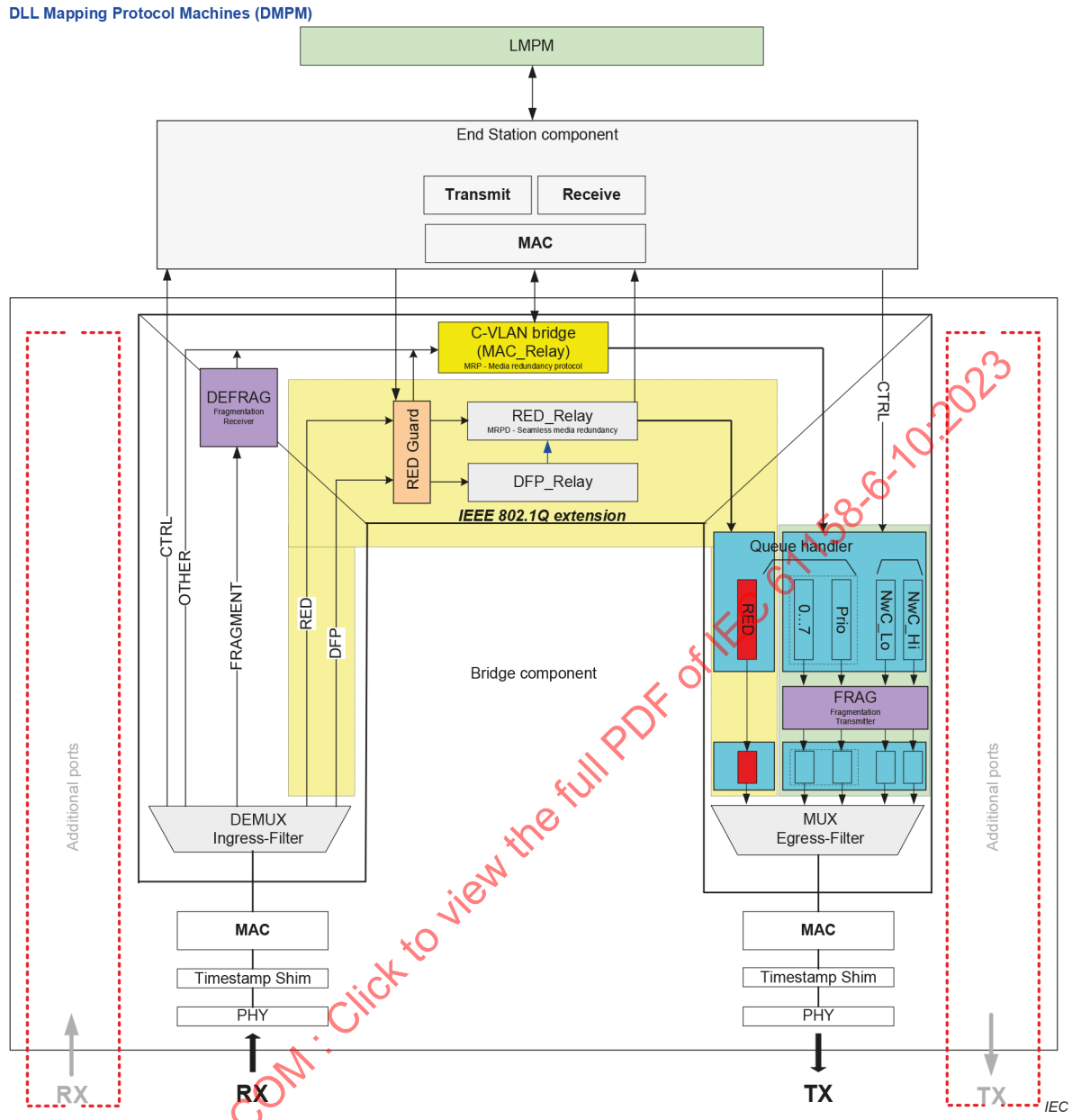


Figure 169 – Structuring of the protocol machines within the DMPM (bridge)

NOTE This kind of diagram is often called baggy pants model.

4.18.2 Data Link Layer Mapping Protocol Machine

4.18.2.1 Primitive definitions

4.18.2.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Data Link Layer Mapping Protocol Machine (LMPM) are described in the service definition and shown in Table 534.

Table 534 – Remote primitives issued or received by LMPM

Primitive	Source	Destination	Associated parameters	Functions
DMUX_P_Data.ind	DEMUX	LMPM	CREP, S_Port, TStamp, DA, SA, N_SDU	Network management protocols are issued directly to the LMPM.
LMPM_A_Data.cnf	LMPM	APMS APMR DCPUCS DCPUCR DCPMCS DCPMCR DCPHMCS DCPHMCR	CREP, LMPM_status	—
LMPM_A_Data.ind	LMPM	APMS APMR DCPUCS DCPUCR DCPMCS DCPMCR DCPHMCS DCPHMCR	CREP, DA, SA, VLANPrio, VLANID, A_SDU	—
LMPM_A_Data.req	APMS APMR DCPUCS DCPUCR DCPMCS DCPMCR DCPHMCS DCPHMCR	LMPM	CREP, DA, SA, VLANPrio, VLANID, A_SDU	—
LMPM_C_Data.cnf	LMPM	SCHEDULER	CREP, LMPM_status	—
LMPM_C_Data.ind	LMPM	CPM	CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status	—
LMPM_C_Data.req	SCHEDULER PPM	LMPM	CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status	—
LMPM_N_Data.cnf	LMPM	APMS APMR IP	CREP, LMPM_status	The placeholder IP is used for other protocol machines.
LMPM_N_Data.ind	LMPM	APMS APMR IP	CREP, DA, SA, VLANPrio, VLANID, N_SDU	The placeholder IP is used for other protocol machines.
LMPM_N_Data.req	APMS APMR IP	LMPM	CREP, DA, SA, VLANPrio, VLANID, N_SDU	The placeholder IP is used for other protocol machines.

Primitive	Source	Destination	Associated parameters	Functions
LMPM_P_Data.cnf	LMPM	PTCP	CREP, D_Port, TStamp, LMPM_status	—
LMPM_P_Data.ind	LMPM	PTCP	CREP, S_Port, TStamp, DA, SA, N_SDU	—
LMPM_P_Data.req	DelayReq DelayRsp MPSM	LMPM	CREP, D_Port, Tstamp, DA, SA, N_SDU	—
MAC_RelayData.cnf	MAC_RELAY	LMPM	CREP, Status	—
MAC_RelayData.ind	MAC_RELAY	LMPM	CREP, DA, SA, LT, VLANPrio, VLANID, C_SDU, APDU_status	—
MAC_RelayData.req	LMPM	MAC_RELAY	CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status	This function is provided by the IEEE Std 802.1Q. It searches the filtering database and executes the M_UNITDATA.req for each found destination port.
MUX_Data.ind	MUX	LMPM	CREP, D_Port, TStamp, Status	Due to the definition of a M_UNITDATA.req an indication is needed to deliver the transmit time stamp if needed
QueueHandler.req	LMPM	QueueHandler	CREP, D_Port, TStamp, N_SDU	For network management protocols or protocols which do not use the filtering database the LMPM accesses the MUX via the QueueHandler.req.
LMPM_R_Data.req	RSII RSIIN RSIR RSIRN	LMPM	CREP, DA, SA, VLANPrio, VLANID, R_SDU	—
LMPM_R_Data.cnf	LMPM	RSII RSIIN RSIR RSIRN	CREP, LMPM_status	—
LMPM_R_Data.ind	LMPM	RSII RSIR RSIRC	CREP, DA, SA, R_SDU	—

4.18.2.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by LMPM are described in the service definition and shown in Table 535.

Table 535 – Local primitives issued or received by LMPM

Primitive	Source	Destination	Associated parameters	Functions
PRO_Protect_req	LMPM	PRO	CREP, DLSDU	Reserved for security
PRO_Protect_cnf (+)	PRO	LMPM	CREP, DLSDU	Reserved for security
PRO_Protect_cnf (-)	PRO	LMPM	CREP, Status	Reserved for security
PRO_Deprotect_req	LMPM	PRO	CREP, DLSDU	Reserved for security
PRO_Deprotect_cnf (+)	PRO	LMPM	CREP, DLSDU	Reserved for security
PRO_Deprotect_cnf (-)	PRO	LMPM	CREP, Status	Reserved for security

4.18.2.2 State transition diagram

The state transition diagram of the LMPM is shown in Figure 170.

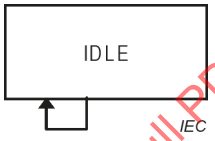


Figure 170 – State transition diagram of LMPM

State of the LMPM is:

IDLE The LMPM is the interface between the bridges or the MAC and the local state machines.

4.18.2.3 State machine description

The LMPM is the interface between MAC and LMPM-User for various data and management services. The services are all handled in the IDLE-State.

The service request will be validated first. A negative validation will result in a negative confirmation. Otherwise, the service will be put into the appropriate priority service queue. Services executed by MAC will be moved from the request queues to the confirmation queues. All valid incoming services will be put into the indication queue.

It implements the access to an optional security layer providing protection (transmit) and deprotection (receive) services.

4.18.2.4 LMPM state table

The LMPM state table is shown in Table 536.

Table 536 – LMPM state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**LMPM_P_Data.req ()** /CheckPDUType () == TimeStamped => QueueHandler.req () //NOTE After the dequeuing and transmitting a MUX_Data.ind () is generated from MUX and issued to the LMPM to convey the transmit Tstamp for the frame	IDLE
2	IDLE	**DMUX_P_Data.ind ()** /CheckPDUType () == TimeStamped => LMPM_P_Data.ind ()	IDLE
3	IDLE	**LMPM_N_Data.req ()** /CheckPDUType () == other => LMPM_N_Data.cnf () MAC_RELAY_Data.req ()	IDLE
4	IDLE	**LMPM_N_Data.req ()** /CheckPDUType () == RPC_UDP => LMPM_N_Data.cnf () MAC_RELAY_Data.req ()	IDLE
5	IDLE	**LMPM_N_Data.req ()** /CheckPDUType () == RPC_UDP, Secure => PRO_Protect_req ()	IDLE
6	IDLE	**PRO_Protect_cnf (+)** /CheckPDUType () == RPC_UDP, Secure => LMPM_N_Data.cnf () MAC_RELAY_Data.req ()	IDLE
7	IDLE	**LMPM_A_Data.req ()** /CheckPDUType () == RTA_CLASS_1 => LMPM_A_Data.cnf () MAC_RELAY_Data.req ()	IDLE
8	IDLE	**LMPM_A_Data.req ()** /CheckPDUType () == RTA_CLASS_1, Secure => PRO_Protect_req ()	IDLE
9	IDLE	**PRO_Protect_cnf (+)** /CheckPDUType () == RTA_CLASS_1, Secure => LMPM_A_Data.cnf () MAC_RELAY_Data.req ()	IDLE
10	IDLE	**LMPM_C_Data.req ()** /CheckPDUType () == RT_CLASS_1 => LMPM_C_Data.cnf () MAC_RELAY_Data.req ()	IDLE
11	IDLE	**LMPM_C_Data.req ()** /CheckPDUType () == RT_CLASS_1, Secure => PRO_Protect_req ()	IDLE

#	Current State	Event /Condition =>Action	Next State
12	IDLE	**PRO_Protect_cnf (+)** /CheckPDUType () == RT_CLASS_1, Secure => LMPM_C_Data.cnf () MAC_RELAY_Data.req ()	IDLE
13	IDLE	**LMPM_C_Data.req ()** /CheckPDUType () == RT_CLASS_3 => LMPM_C_Data.cnf () QueueHandler.req ()	IDLE
14	IDLE	**LMPM_C_Data.req ()** /CheckPDUType () == RT_CLASS_STREAM => LMPM_C_Data.cnf () MAC_RELAY_Data.req ()	IDLE
15	IDLE	**LMPM_C_Data.req ()** /CheckPDUType () == RT_CLASS_STREAM, Secure => PRO_Protect_req ()	IDLE
16	IDLE	**PRO_Protect_cnf (+)** /CheckPDUType () == RT_CLASS_STREAM, Secure => LMPM_C_Data.cnf () MAC_RELAY_Data.req ()	IDLE
17	IDLE	**LMPM_R_Data.req ()** /CheckPDUType () == RSI => LMPM_R_Data.cnf () MAC_RELAY_Data.req ()	IDLE
18	IDLE	**LMPM_R_Data.req ()** /CheckPDUType () == RSI, Secure => PRO_Protect_req ()	IDLE
19	IDLE	**PRO_Protect_cnf (+)** /CheckPDUType () == RSI, Secure => LMPM_R_Data.cnf () MAC_RELAY_Data.req ()	IDLE
20	IDLE	**RED_RELAY_Data.ind ()** /CheckPDUType () == RT_CLASS_3 => LMPM_C_Data.ind ()	IDLE
21	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RT_CLASS_1 => LMPM_C_Data.ind ()	IDLE
22	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RT_CLASS_1, Secure => PRO_Deprotect_req ()	IDLE
23	IDLE	**PRO_Deprotect_cnf (+)** /CheckPDUType () == RT_CLASS_1, Secure => LMPM_C_Data.ind ()	IDLE
24	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RT_CLASS_STREAM => LMPM_C_Data.ind ()	IDLE
25	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RT_CLASS_STREAM, Secure => PRO_Deprotect_req ()	IDLE

#	Current State	Event /Condition =>Action	Next State
26	IDLE	**PRO_Deprotect_cnf (+)** /CheckPDUType () == RT_CLASS_STREAM, Secure => LMPM_C_Data.ind ()	IDLE
27	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RTA_CLASS_1 => LMPM_A_Data.ind ()	IDLE
28	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RTA_CLASS_1, Secure => PRO_Deprotect_req ()	IDLE
29	IDLE	**PRO_Deprotect_cnf (+)** /CheckPDUType () == RTA_CLASS_1, Secure => LMPM_A_Data.ind ()	IDLE
30	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RSI => LMPM_R_Data.ind ()	IDLE
31	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RSI, Secure => PRO_Deprotect_req ()	IDLE
32	IDLE	**PRO_Deprotect_cnf (+)** /CheckPDUType () == RSI, Secure => LMPM_R_Data.ind ()	IDLE
33	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RPC_UDP => LMPM_N_Data.ind ()	IDLE
34	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == RPC_UDP, Secure => PRO_Deprotect_req ()	IDLE
35	IDLE	**PRO_Deprotect_cnf (+)** /CheckPDUType () == RPC_UDP, Secure => LMPM_N_Data.ind ()	IDLE
36	IDLE	**MAC_RELAY_Data.ind ()** /CheckPDUType () == other => LMPM_N_Data.ind ()	IDLE
37	IDLE	**PRO_Deprotect_cnf (-)** => ignore //NOTE Unknown frame, Invalid deprotection, discard received frame	IDLE
38	IDLE	**PRO_Protect_cnf (-)** => ignore //NOTE Unknown frame, Protection was not possible, discard to be transmitted frame	IDLE
39	IDLE	**MUX_Data.ind ()** => LMPM_P_Data.cnf ()	IDLE
40	IDLE	**MAC_RELAY_Data.cnf ()** => ignore	IDLE

4.18.2.5 Functions, Macros, Timers and Variables

Table 537 contains the functions, macros, timers, and variables used by the LMPM and their arguments and their descriptions.

Table 537 – Functions, Macros, Timers and Variables used by the LMPM

Name	Type	Function/Meaning
CheckPDUType	Function	This local function checks the PDU type and returns the values – RT_CLASS_1 (– RT_CLASS_UDP) – RT_CLASS_3 – RT_CLASS_STREAM – RTA_CLASS_1 (– RTA_CLASS_UDP) – RPC_UDP – RSI – Other – TimeStamped This local functions returns "Secure" as the second value if the frame shall be protected or deprotected. Information contained in the frame together with AREP, CREP and protocol related fields are evaluated to identify the status of a frame.

4.19 Void

Intentionally left blank.

4.20 Additional information

Annex A, Annex B, Annex C, Annex D, Annex E, Annex F, Annex G, Annex H, Annex I, Annex J, Annex K, Annex L, Annex M, Annex N, Annex O, Annex P, Annex Q, Annex R, Annex S, Annex T, Annex U, Annex V, Annex W, Annex X and Annex Y give additional information.

5 Application layer protocol specification for distributed I/O

5.1 FAL syntax description

5.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

5.1.2 APDU abstract syntax

The abstract syntax of the Application Layer PDUs referred to as APDUs is defined in Table 538, Table 539 , Table 540, Table 541, Table 542, and shall represent the content of the DLSDU in Table 10, Table 11, and Table 12. The defined order of octets shall be used to convey the APDUs.

An IO device or IO controller application:

– shall support the highest BlockVersionHigh and/or BlockVersionLow shown in Table 538, Table 539, Table 540, Table 541 and Table 542, and

– may support all BlockVersionHigh and/or BlockVersionLow specified in this or earlier versions of this specification.

Table 538 – IO APDU substitutions

Substitution name	Structure
RTA-SDU	AlarmNotification-PDU ^ AlarmAck-PDU
BlockHeader	BlockType, BlockLength, BlockVersionHigh, BlockVersionLow
AlarmNotification-PDU	BlockHeader, AlarmType, API [a], SlotNumber [a], SubslotNumber [a], ModuleIdentNumber, SubmoduleIdentNumber, AlarmSpecifier, AlarmPayload Special case “Alarm transport container” (see Table 555): BlockHeader, AlarmType, API(=0), SlotNumber(=0x8000), SubslotNumber(=0), ModuleIdentNumber(=0), SubmoduleIdentNumber(=0), AlarmSpecifier, AlarmPayload(=RS_AlarmItem) [a] These fields address the submodule which can be source or sink of an alarm. They contain the source if the IO device issues an alarm and the sink if the IO controller issues an alarm. The BlockType field within the BlockHeader shall be set to AlarmNotification high or low according to Table 543.
AlarmAck-PDU	BlockHeader, AlarmType, API, SlotNumber, SubslotNumber, AlarmSpecifier, PNIOStatus The BlockType field within the BlockHeader shall be set to AlarmAck high or low according to Table 543. Special case “No Error”: PNIOStatus(=0x00, 0x00, 0x00, 0x00) Special case “Alarm Type Not Supported” or if a reserved Alarm Type is used: PNIOStatus(=0xDA, 0x81, 0x3C, 0x00) Special case “Wrong Submodule State”: PNIOStatus(=0xDA, 0x81, 0x3C, 0x01) Special case “IOCARSR: Backup – Alarm not executed”: PNIOStatus(=0xDA, 0x81, 0x3C, 0x02)
AlarmPayload	[MaintenanceItem], [DiagnosisItem] ^ [AlarmItem] ^ [Upload&RetrievalItem] ^ [iParameterItem] ^ [PE_AlarmItem] ^ [RS_AlarmItem] ^ [PRAL_AlarmItem] See Table 547
MaintenanceItem	UserStructureIdentifier, BlockHeader, Padding, Padding, MaintenanceStatus
Upload&RetrievalItem	UserStructureIdentifier, BlockHeader, Padding, Padding, (URRecordIndex, URRecordLength)* Special case “Retrieve all stored records”: UserStructureIdentifier, BlockHeader(=0x0F04), Padding, Padding, URRecordIndex(=0), URRecordLength(=0)
iParameterItem	UserStructureIdentifier, BlockHeader, Padding, Padding, iPar_Req_Header, Max_Segm_Size, Transfer_Index, Total_iPar_Size
DiagnosisItem	UserStructureIdentifier, ChannelDiagnosisData* ^ ExtChannelDiagnosisData* ^ QualifiedChannelDiagnosisData* ^ ManufacturerData* Special case ChannelDiagnosisData: UserStructureIdentifier(=0x8000), ChannelDiagnosisData* Special case ExtChannelDiagnosisData: UserStructureIdentifier(=0x8002), ExtChannelDiagnosisData* Special case QualifiedChannelDiagnosisData: UserStructureIdentifier(=0x8003), QualifiedChannelDiagnosisData* Special case ManufacturerData: UserStructureIdentifier(=0x0000 – 0x7FFF), ManufacturerData*
PE_AlarmItem	UserStructureIdentifier(=PE_EnergySavingStatus), BlockHeader, PE_OperationalMode

Substitution name	Structure
RS_AlarmItem	UserStructureIdentifier, RS_AlarmInfo ^ RS_EventInfo Special case RS_LowWatermark: UserStructureIdentifier(=0x8300), RS_AlarmInfo Special case RS_Timeout: UserStructureIdentifier(=0x8301), RS_AlarmInfo Special case RS_Overflow: UserStructureIdentifier(=0x8302), RS_AlarmInfo Special case RS_Event: UserStructureIdentifier(=0x8303), RS_EventInfo
PRAL_AlarmItem	UserStructureIdentifier(=0x8320), ChannelNumber, PRAL_ChannelProperties, PRAL_Reason, PRAL_ExtReason, PRAL_ReasonAddValue
AlarmItem	UserStructureIdentifier, Data*
PROFINETIOServiceReqPDU	IODConnectReq ^ IODWriteReq ^ IODWriteMultipleReq ^ IODReadReq ^ IODControlReq ^ IODReleaseReq ^ IOXControlReq ^ IOXSecureReq
PROFINETIOServiceResPDU	IODConnectRes ^ IODWriteRes ^ IODWriteMultipleRes ^ IODReadRes ^ IODControlRes ^ IODReleaseRes ^ IOXControlRes ^ IOXSecureRsp
IODConnectReq with StartupMode:=1	ARBlockReq, {[IOCRBlockReq*], [AlarmCRBlockReq], [ExpectedSubmoduleBlockReq*] [b], [PrmServerBlock], [MCRBlockReq*] [a], [ARRPCBlockReq], [IRInfoBlock], [SRInfoBlock], [ARVendorBlockReq*], [ARFSUBlock], [RSInfoBlock], [ARAlgorithmInfoBlock] [f] } Special case: “IOCARSingle” or “IOSAR”: ARBlockReq, {IOCRBlockReq*, AlarmCRBlockReq, ExpectedSubmoduleBlockReq*, [PrmServerBlock], [MCRBlockReq*] [a], [ARRPCBlockReq], [IRInfoBlock] [d], [ARVendorBlockReq*], [ARFSUBlock], [RSInfoBlock], [ARAlgorithmInfoBlock] [f] } Special case: “IOSAR with ARProperties.DeviceAccess:=1”: ARBlockReq [c], [ARAlgorithmInfoBlock] [f] Special case: “IOCARSR”: ARBlockReq, {IOCRBlockReq*, AlarmCRBlockReq, ExpectedSubmoduleBlockReq*, [PrmServerBlock], [MCRBlockReq*] [a], [ARRPCBlockReq], SRInfoBlock, [ARVendorBlockReq*], [RSInfoBlock] , [ARAlgorithmInfoBlock] [f] } Special case: “Secure Association Control”: ARBlockReq, [IOCRBlockReq*], [AlarmCRBlockReq], [MCRBlockReq*] [a], [ARRPCBlockReq] [e], ARAlgorithmInfoBlock [a] The field MCRBlockReq shall only be present if at least one IOCRBlockReq contains the value MULTICAST_CONSUMER_CR as an IO CR Type. [b] The field ExpectedSubmoduleBlockReq shall only contain submodules which are referred in at least one IOCRBlockReq. [c] The field CMInitiatorActivityTimeoutFactor specifies the watchdog time. [d] The field IRInfoBlock shall be part of the IODConnect.req if IOCRProperties.RTClass == 0x03 is used. [e] The field ARRPCBlockReq shall only be present if RPC is used. [f] The ARAlgorithmInfoBlock block shall only be present if ARProperties.ProtectionProperties != 0.

Substitution name	Structure
IODConnectRes with StartupMode:=1	ARBlockRes, {[IOCRBlockRes*], [AlarmCRBlockRes], [ModuleDiffBlock], [ARRPCBlockRes] [a], ARServerBlockRes, [ARVendorBlockRes*] [b] } Special case: “IOCARSingle” or “IOSAR”: ARBlockRes, {IOCRBlockRes*, AlarmCRBlockRes, [ModuleDiffBlock], [ARRPCBlockRes] [a], ARServerBlockRes, [ARVendorBlockRes*] [b] } Special case: “IOSAR with ARProperties.DeviceAccess=1”: ARBlockRes, ARServerBlockRes Special case: “Secure Association Control”: ARBlockRes, [ARRPCBlockRes] [a] [a] The field ARRPCBlockRes shall only be present if IODConnectReq contains an ARRPCBlockReq. The BlockVersionLow:=1 shall be used if ARProperties.ProtectionProperties != 0. [b] The number of ARVendorBlockRes entries shall be equal to the number of ARVendorBlockReq entries in the IODConnectReq.
IODConnectReq with StartupMode:=Legacy	ARBlockReq, {[IOCRBlockReq*], [AlarmCRBlockReq], [ExpectedSubmoduleBlockReq*] [b], [PrmServerBlock], [MCRBlockReq*] [a], [ARRPCBlockReq] } Special case: “IOCARSingle” or “IOSAR”: ARBlockReq, {IOCRBlockReq*, AlarmCRBlockReq, ExpectedSubmoduleBlockReq*, [PrmServerBlock], [MCRBlockReq*] [a], [ARRPCBlockReq] } Special case: “IOSAR with ARProperties.DeviceAccess:=1”: ARBlockReq [c] Special case: “IOCARSingle using RT_CLASS_3: ARBlockReq, {IOCRBlockReq* [d], AlarmCRBlockReq, ExpectedSubmoduleBlockReq*, [PrmServerBlock], [MCRBlockReq*] [a], [ARRPCBlockReq] } [a] The field MCRBlockReq shall only be present if at least one IOCRBlockReq contains the value MULTICAST_CONSUMER_CR as an IO CR Type. [b] The field ExpectedSubmoduleBlockReq shall only contain submodules which are referred in at least one IOCRBlockReq. [c] The field CMInitiatorActivityTimeoutFactor specifies the watchdog time. [d] To establish an “IOCARSingle using RT_CLASS_3” with RT_CLASS_3 input and output communication relations, one input and one output communication relation shall exist.
IODConnectRes with StartupMode:=Legacy	ARBlockRes, {[IOCRBlockRes*], [AlarmCRBlockRes], [ModuleDiffBlock], [ARRPCBlockRes] [a] } Special case: “IOCARSingle” or “IOSAR”: ARBlockRes, {IOCRBlockRes*, AlarmCRBlockRes, [ModuleDiffBlock], [ARRPCBlockRes] [a] } Special case: “IOSAR with ARProperties.DeviceAccess=1”: ARBlockRes [a] The field ARRPCBlockRes shall only be present if IODConnectReq contains an ARRPCBlockReq.
IODWriteMultipleReq	IODWriteReqHeader [a], (IODWriteReq, [Padding*] [b])* [a] The value of the parameter index shall be 0xE040. [b] The number of padding octets (value=0) shall be 0,1,2,3 to have 32 bit alignment to the next IODWriteReqHeader.
IODWriteReq	IODWriteReqHeader, RecordDataWrite

Substitution name	Structure
IODWriteReqHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* [a], Index, RecordDataLength, RWPadding* [b] [a] The number of padding octets (value=0) in the write request is 2. [b] The number of padding octets (value=0) in the write request is 24.
IODWriteRes	IODWriteResHeader [a] [a] The value of the parameter PNIOStatus shall be (0,0,0,0) or a duplicate of the PNIOStatus from the NDRDataResponse.
IODWriteMultipleRes	IODWriteResHeader [a] [b], (IODWriteResHeader)* [c] [a] The value of the parameter index shall be 0xE040. [b] The value of the parameter PNIOStatus shall be (0,0,0,0) or a duplicate of the PNIOStatus from the NDRDataResponse. [c] The value of the parameter PNIOStatus contains the result of the particular write.
IODWriteResHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* [a], Index, RecordDataLength, AdditionalValue1, AdditionalValue2, PNIOStatus, RWPadding* [b] [a] The number of padding octets (value=0) in the write response is 2. [b] The number of padding octets (value=0) in the write response is 16.
IODReadReq	IODReadReqHeader, [RecordDataReadQuery]
IODReadReqHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* [a], Index, RecordDataLength, TargetARUUID [b] ^ RWPadding* [c], RWPadding* [d] Special case “Access through CIM interface”: BlockHeader, SeqNumber(0), ARUUID(0), API(0), SlotNumber(0), SubslotNumber(0), Padding* [a], Index, RecordDataLength, RWPadding* [c], RWPadding* [d] [a] The number of padding octets (value=0) in the read request is 2. [b] The optional field TargetARUUID shall only be used in conjunction with the value 0 of ARUUID (Implicit AR). [c] The number of padding octets (value=0) in the read request is 16. [d] The number of padding octets (value=0) in the read request is 8.
IODReadReqHeader with BlockVersionLow:=1	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* [a], Index, RecordDataLength, TargetARUUID [b] ^ RWPadding* [c], RequestedBlockVersion, RWPadding* [d] Special case “Access through CIM interface”: BlockHeader, SeqNumber(0), ARUUID(0), API(0), SlotNumber(0), SubslotNumber(0), Padding* [a], Index, RecordDataLength, RWPadding* [c], RequestedBlockVersion, RWPadding* [d] [a] The number of padding octets (value=0) in the read request is 2. [b] The optional field TargetARUUID shall only be used in conjunction with the value 0 of ARUUID (Implicit AR). [c] The number of padding octets (value=0) in the read request is 16. [d] The number of padding octets (value=0) in the read request is 6.
RequestedBlockVersion	BlockVersionHigh, BlockVersionLow Special case “Provide responders newest version”: BlockVersionHigh(0), BlockVersionLow(0)
IODReadRes	IODReadResHeader, RecordDataRead

Substitution name	Structure
IODReadResHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* [a], Index, RecordDataLength, AdditionalValue1, AdditionalValue2, RWPadding* [b] [a] The number of padding octets (value=0) in the read res is 2. [b] The number of padding octets (value=0) in the read res is 20.
IODControlReq	ControlBlockConnect [a] ^ (ControlBlockConnect, SubmoduleListBlock) [b] ^ ControlBlockPlug [c] [a] Shall be used in the context of a connect or a PrmBegin/PrmEnd sequence for ControlCommand.PrmEnd [b] Shall be used in the context of a PrmBegin/PrmEnd sequence for ControlCommand.PrmBegin with SubmoduleListBlock [c] Shall be used in the context of a plug/release for ControlCommand.PrmEnd
IODControlRes	ControlBlockConnect [a] ^ (ControlBlockConnect, [ModuleDiffBlock]) [b] ^ ControlBlockPlug [c] ^ NULL [d] [a] Shall be used in the context of a connect or a PrmBegin/PrmEnd sequence for ControlCommand.PrmEnd [b] Shall be used in the context of a PrmBegin/PrmEnd sequence for ControlCommand.PrmBegin if the checking of the SubmoduleListBlock leads to a ModuleDiffBlock [c] Shall be used in the context of a plug/release for ControlCommand.PrmEnd [d] In case of a negative response NULL shall be transmitted.
IODReleaseReq	ReleaseBlock
IODReleaseRes	ReleaseBlock ^ NULL [a] [a] In case of a negative response NULL shall be transmitted.
IOXControlReq with StartupMode:=1	(ControlBlockConnect, [ModuleDiffBlock]) [a] ^ (ControlBlockPlug, [ModuleDiffBlock]) [b] [a] Shall be used in the context of a connect or a PrmBegin/PrmEnd sequence for ControlCommand.ApplicationReady with ModuleDiffBlock if needed. [b] Shall be used in the context of a plug/release for ControlCommand.ApplicationReady with ModuleDiffBlock if needed. The field ModuleDiffBlock shall only be present if it contains entries. An empty ModuleDiffBlock shall be omitted.
IOXControlRes with StartupMode:=1	ControlBlockConnect [a] ^ ControlBlockPlug [b] ^ NULL [c] [a] Shall be used in the context of a connect or a PrmBegin/PrmEnd sequence for ControlCommand.ApplicationReady [b] Shall be used in the context of a plug/release for ControlCommand.ApplicationReady [c] In case of a negative response NULL shall be transmitted.
IOXControlReq with StartupMode:=Legacy	(ControlBlockConnect, [ModuleDiffBlock]) [a] ^ (ControlBlockPlug, [ModuleDiffBlock]) [b] ^ ControlBlockRFC [c] ^ ControlBlockRTC [d] [a] Shall be used in the context of a connect for ControlCommand.ApplicationReady with ModuleDiffBlock if needed. [b] Shall be used in the context of a plug/release for ControlCommand.ApplicationReady with ModuleDiffBlock if needed. [c] Shall be used for ControlCommand.ReadyForCompanion [d] Shall be used for ControlCommand.ReadyForRT_CLASS_3 The field ModuleDiffBlock shall only be present if it contains entries. An empty ModuleDiffBlock shall be omitted.

Substitution name	Structure
IOXControlRes with StartupMode:=Legacy	ControlBlockConnect [a] ^ ControlBlockPlug [b] ^ ControlBlockRFC [c] ^ ControlBlockRTC [d] ^ NULL [e] [a] Shall be used in the context of a connect for ControlCommand.ApplicationReady [b] Shall be used in the context of a plug/release for ControlCommand.ApplicationReady [c] Shall be used for ControlCommand.ReadyForCompanion [d] Shall be used for ControlCommand.ReadyForRT_CLASS_3 [e] In case of a negative response NULL shall be transmitted.
ARBlockReq	BlockHeader, ARType, ARUUID, SessionKey, CMInitiatorMacAdd, CMInitiatorObjectUUID, ARProperties, CMInitiatorActivityTimeoutFactor, InitiatorUDPRTPort [a], StationNameLength, CMInitiatorStationName [b] [a] If no RT_CLASS_UDP CR is used, the value for InitiatorUDPRTPort shall be set to 0x8892. [b] This coding leads to a misalignment if the length of CMInitiatorStationName is uneven. An IO device should support a following [Padding*] to ensure Unsigned32 alignment. The BlockType field within the BlockHeader shall be set to ARBlockReq according to Table 543.
ARBlockRes	BlockHeader, ARType, ARUUID, SessionKey, CMResponderMacAdd, ResponderUDPRTPort [a] [a] If no RT_CLASS_UDP CR is used, the value for ResponderUDPRTPort shall be set to 0x8892. The BlockType field within the BlockHeader shall be set to ARBlockRes according to Table 543.

Substitution name	Structure
IOCRBlockReq	BlockHeader, IOCRType, IOCRReference, LT, IOCRProperties, DataLength, FrameID, SendClockFactor, ReductionRatio, Phase, Sequence, FrameSendOffset [a], DataHoldFactor [b], DataHoldFactor, IOCRTagHeader [d], IOCRMulticastMACAdd [c], NumberOfAPIs, (API, NumberOfIODataObjects, (SlotNumber, SubslotNumber, IODataObjectFrameOffset)*, NumberOfIOCS, (SlotNumber, SubslotNumber, IOCSFrameOffset)*)* [a] The content of this field shall be consistent with the corresponding field in PDIRFrameData for RT_CLASS_3. [b] This field exists due to legacy reasons and was formerly called WatchDogFactor. [c] This field contains for Time-aware streams, class HIGH and LOW, the stream destination MACAddress used by the corresponding PPM. [d] This field contains for Time-aware streams, class HIGH and LOW, and Streams, class RT, a reference to the to be used TCI.PCP and TCI.VID. The BlockType field within the BlockHeader shall be set to IOCRBlockReq according to Table 543. Special Case IOCRType == OUTPUT_CR not using RT_CLASS_3 The field FrameID shall not be evaluated. Special Case IOCRType == OUTPUT_CR using RT_CLASS_3 The field FrameID shall contain the FrameID used in the corresponding PDIRData. Special Case IOCRType == MULTICAST_CONSUMER_CR The field FrameID shall contain a FrameID from the defined multicast range. The configuration tool shall guarantee that this FrameID is unique within the project context. Special Case Time-aware streams, class HIGH and LOW, and Streams, class RT, and IOCRType == OUTPUT_CR The field FrameID shall not be evaluated. Special Case Time-aware streams, class HIGH and LOW, and Streams, class RT, and IOCRType == INPUT_CR The field FrameID shall be calculated by the CPM and used by the corresponding PPM. Special Case Time-aware streams, class HIGH and LOW, and Streams, class RT, and IOCRType == MULTICAST_PROVIDER_CR The field FrameID shall be calculated by the PPM and used by the corresponding CPM. Special Case Time-aware streams, class HIGH and LOW, and Streams, class RT, and IOCRType == MULTICAST_CONSUMER_CR The field FrameID shall be used by the corresponding CPM.
MCRBlockReq	BlockHeader, IOCRReference, AddressResolutionProperties, MCITimeoutFactor, StationNameLength, ProviderStationName, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. The BlockType field within the BlockHeader shall be set to MCRBlockReq according to Table 543.
IOCRBlockRes	BlockHeader, IOCRType, IOCRReference, FrameID The BlockType field within the BlockHeader shall be set to IOCRBlockRes according to Table 543.

Substitution name	Structure
ExpectedSubmoduleBlockReq	BlockHeader, NumberOfAPIs, (API, SlotNumber [a], ModuleIdentNumber, ModuleProperties, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber, SubmoduleProperties [b], (DataDescription, SubmoduleDataLength, LengthIOPS, LengthIOCS)*)*)* [a] This parameter is constant for one ExpectedSubmoduleBlockReq because each contains the data for one slot only. [b] The field SubmoduleProperties.Type determines the number of subsequent data description blocks. The BlockType field within the BlockHeader shall be set to ExpectedSubmoduleBlockReq according to Table 543.
ModuleDiffBlock	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber [a], ModuleState [b], NumberOfSubmodules, [(SubslotNumber, SubmoduleIdentNumber [c], SubmoduleState)*])*)* [a] Real identification data; should be zero in case of ModuleState := NoModule [b] If ModuleState := NoModule then NumberOfSubmodules shall be zero. The subsequent part shall be omitted. For all other ModuleState only the SubmoduleState shall be used to decide whether the submodule shall be parameterized. [c] Real identification data; should be zero in case of SubmoduleState.IdentInfo:= NoSubmodule The BlockType field within the BlockHeader shall be set to ModuleDiffBlock according to Table 543.
AlarmCRBlockReq	BlockHeader, AlarmCRType, LT, AlarmCRProperties, RTATimeoutFactor, RTARetries, LocalAlarmReference, MaxAlarmDataLength, AlarmCRTagHeaderHigh, AlarmCRTagHeaderLow The BlockType field within the BlockHeader shall be set to AlarmCRBlockReq according to Table 543.
AlarmCRBlockRes	BlockHeader, AlarmCRType, LocalAlarmReference, MaxAlarmDataLength The BlockType field within the BlockHeader shall be set to AlarmCRBlockRes according to Table 543.
PrmServerBlock	BlockHeader, ParameterServerObjectUUID, ParameterServerProperties, CMInitiatorActivityTimeoutFactor, StationNameLength, ParameterServerStationName [a] [a] An IO device should support a following [Padding*] to ensure Unsigned32 alignment. The BlockType field within the BlockHeader shall be set to PrmServerBlock according to Table 543.
ARRPCBlockReq	BlockHeader, InitiatorRPCServerPort The BlockType field within the BlockHeader shall be set to ARRPCBlockReq according to Table 543.
ARRPCBlockRes with BlockVersionLow:=0	BlockHeader, ResponderRPCServerPort The BlockType field within the BlockHeader shall be set to ARRPCBlockRes according to Table 543.
ARRPCBlockRes with BlockVersionLow:=1	BlockHeader, ResponderRPCServerPort, ResponderActivityUUID [a] [a] Used in conjunction with ARProperties.ProtectionProperties The BlockType field within the BlockHeader shall be set to ARRPCBlockRes according to Table 543.
ARServerBlockRes	BlockHeader, StationNameLength, CMResponderStationName, [Padding* [a]] [a] The number of Padding octets shall ensure Unsigned32 alignment. The BlockType field within the BlockHeader shall be set to ARServerBlockRes according to Table 543.

Substitution name	Structure
IRInfoBlock	BlockHeader, Padding* [a], IRDataUUID, Padding [a], NumberOfIOCRs [b], [(IOCRReference, SubframeOffset, SubframeData)*] [a] The number of Padding octets shall be 2. [b] This field shall be coded as zero if no DFP is used. The BlockType field within the BlockHeader shall be set to IRInfoBlock according to Table 543.
SRInfoBlock	BlockHeader, RedundancyDataHoldFactor, SRProperties The BlockType field within the BlockHeader shall be set to SRInfoBlock according to Table 543.
ARFSUBlock	ARFSUDataAdjust The BlockType field within the BlockHeader shall be set to ARFSUBlock according to Table 543.
RSInfoBlock	BlockHeader, [Padding* [a]], RS_Properties [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. The BlockType field within the BlockHeader shall be set to RSInfoBlock according to Table 543.
ARAlgorithmInfoBlock	Reserved for security The BlockType field within the BlockHeader shall be set to ARAlgorithmInfoBlock according to Table 543.
ARVendorBlockReq	BlockHeader, APStructureIdentifier, API, [Data*], [Padding* [a]] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. The BlockType field within the BlockHeader shall be set to ARVendorBlockReq according to Table 543. Special case “Extended identification rules” BlockHeader, APStructureIdentifier (:=0x8000), API (:=0), ExpectedExtended-IdentificationInfo, [Padding* [a]]
ARVendorBlockRes	BlockHeader, APStructureIdentifier, API, [Data*], [Padding* [a]] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. The BlockType field within the BlockHeader shall be set to ARVendorBlockRes according to Table 543. Special case “Extended identification rules” BlockHeader, APStructureIdentifier (:=0x8000), API (:=0), RealExtended-IdentificationInfo, [Padding* [a]]
ExpectedExtended-IdentificationInfo	ExtendedIdentificationInfo
RealExtendedIdentificationInfo	ExtendedIdentificationInfo
ExtendedIdentificationInfo	ExtendedIdentificationUUID, ExtendedIdentificationVersionHigh, ExtendedIdentificationVersionLow
SubframeBlock	BlockHeader, FrameID [a], SFIOCRProperties, SubframeData* [b] [a] For MRPD only the SubframeBlock for the FrameID with the least significant bit set to zero shall be created. [b] The ordering of SubframeData octets shall be used for the non-redundant path The BlockType field within the BlockHeader shall be set to SubframeBlock according to Table 543.
PDIRSubframeData	BlockHeader, NumberOfSubframeBlocks, SubframeBlock* The BlockType field within the BlockHeader shall be set to PDIRSubframeBlock according to Table 543.

Substitution name	Structure
ControlBlockConnect	BlockHeader, Padding, Padding, ARUUID, SessionKey, Padding, Padding, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to IODBlockReq, IODBlockRes, IOXBlockReq and IOXBlockRes according to Table 543.
ControlBlockPlug	BlockHeader, Padding, Padding, ARUUID, SessionKey, AlarmSequenceNumber, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to IOXBlockReq, IOXBlockRes according to Table 543. The field AlarmSequenceNumber shall contain the value of the sub-field AlarmSpecifier.SequenceNumber of the corresponding AlarmNotification-PDU.
ControlBlockRFC	BlockHeader, Padding, Padding, ARUUID, SessionKey, Padding, Padding, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to ReadyForCompanionBlock according to Table 543.
ControlBlockRTC	BlockHeader, Padding, Padding, ARUUID, SessionKey, Padding, Padding, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to ReadyForRTCLASS3Block according to Table 543.
ReleaseBlock	BlockHeader, Padding, Padding, ARUUID, SessionKey, Padding, Padding, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to ReleaseBlock according to Table 543.
SubmoduleListBlock	BlockHeader, NumberOfEntries, (API, SlotNumber, SubslotNumber)* The BlockType field within the BlockHeader shall be set to SubmoduleListBlock according to Table 543.
RecordDataReadQuery	UNIServiceReq ^ CIMSecurityServiceReq ^ CIMDCPServiceReq ^ UploadBLOBQuery ^ (BlockHeader, Data* [a]) [a] The usage, structure and values shall be defined for: 1) Manufacturer specific index range by the manufacturer 2) Profile specific index ranges by the profiles 3) All other index ranges by this document
RecordDataRead	DiagnosisData* ^ ExpectedIdentificationData* ^ RealIdentificationData ^ SubstituteValue ^ RecordInputDataObjectElement ^ RecordOutputDataObjectElement ^ ARData ^ IMData ^ LogBookData ^ ModuleDiffBlock ^ APIData ^ PDPortDataAdjust ^ PDPortDataCheck* ^ PDPortDataReal* ^ PDPortDataRealExtended ^ PDIRData ^ PDSyncData ^ PDTimeData ^ PdevData ^ IsochronousModeData ^ PDInterfaceAdjust ^ PDInterfaceMrpDataAdjust ^ PDInterfaceMrpDataCheck ^ PDInterfaceMrpDataReal ^ PDPortMrpDataAdjust ^ PDPortMrpDataReal ^ PDPortFODataReal ^ PDPortFODataAdjust ^ PDPortFODataCheck ^ PDRealData* ^ PDExpectedData* ^ PDNCDataCheck ^ PDPortStatistic ^ I&M0FilterData ^ ARFSUDataAdjust ^ PDInterfaceFSUDataAdjust ^ PDInterfaceDataReal ^ PDIRSubframeData ^ PE_EntityFilterData ^ PE_EntityStatusData ^ AssetManagementData ^ RS_AdjustObserver ^ RS_GetEvent ^ PDPortMrpIcDataAdjust ^ PDPortMrpIcDataCheck ^ PDPortMrpIcDataReal ^ PDPortSFPDataCheck ^ ApplicationReadyBlock ^ CIMDataRead ^ PDRsiInstances ^ UNIServiceRsp ^ CIMSecurityServiceRsp ^ CIMDCPServiceRsp ^ UploadBLOB ^ Data* [b] ^ NULL [a] [a] NULL shall be used if a requested well-known data record is empty (for example Diagnosis, ModuleDiffBlock, ARData, ...). [b] Manufacturer specific data

Substitution name	Structure
RecordDataWrite	SubstituteValue ^ IMDataWrite ^ PDPortDataAdjust ^ PDPortDataCheck ^ PDIRData ^ PDSyncData ^ PDTimeData ^ IsochronousModeData ^ PDInterfaceMrpDataAdjust ^ PDInterfaceMrpDataCheck ^ PDPortMrpDataAdjust ^ PDPortFODataAdjust ^ PDPortFODataCheck ^ PDNCDataCheck ^ ARFSUDataAdjust ^ PDInterfaceFSUDataAdjust ^ PDIRSubframeData ^ PDInterfaceAdjust ^ CombinedObjectContainer ^ RS_AdjustObserver ^ RS_AckEvent ^ PDPortMrpIcDataAdjust ^ PDPortMrpIcDataCheck ^ PDPortSFPDataCheck ^ CIMDataWrite ^ Data* [a] [a] Manufacturer specific data
DiagnosisData with BlockVersionLow=0	BlockHeader, ChannelDiagnosis ^ ManufacturerSpecificDiagnosis ^ ExtChannelDiagnosis BlockVersionLow=0 shall be supported by IO controller and IO supervisor. It shall not be generated by IO device.
DiagnosisData with BlockVersionLow=1	BlockHeader, API, ChannelDiagnosis ^ ManufacturerSpecificDiagnosis ^ ExtChannelDiagnosis ^ QualifiedChannelDiagnosis BlockVersionLow=1 shall be generated by IO device and support by IO controller and IO supervisor.
ChannelDiagnosis	SlotNumber, SubslotNumber, ChannelNumber(0x8000), ChannelProperties [a], UserStructureIdentifier(0x8000), ChannelDiagnosisData* [a] The field ChannelProperties.Type, the field ChannelProperties.Direction, the field ChannelProperties.Maintenance shall be set to zero. The field ChannelProperties.Specifier shall be set to appear if at least one ChannelProperties.Specifier in the ChannelDiagnosisData is set to appear in conjunction with ChannelProperties.Maintenance(=Fault). Else, the field ChannelProperties.Specifier shall be set to disappear.
ChannelDiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType
ManufacturerSpecificDiagnosis	SlotNumber, SubslotNumber, ChannelNumber(0x8000), ChannelProperties [a] [b] [c], UserStructureIdentifier, ManufacturerData* Special case: ManufacturerSpecificDiagnosis together with AlarmNotification and DiagnosisData [a] The field ChannelProperties.Type, the field ChannelProperties.Direction, the field ChannelProperties.Accumulative shall be set to zero, the field ChannelProperties.Maintenance shall be set to Fault. The field ChannelProperties.Specifier shall be set to appear. Special case: ManufacturerSpecificDiagnosis together with AlarmNotification (no DiagnosisData) [b] The field ChannelProperties.Type, the field ChannelProperties.Direction, the field ChannelProperties.Accumulative shall be set to zero, the field ChannelProperties.Maintenance shall be set to Fault. The field ChannelProperties.Specifier shall be set to disappear. Special case: ManufacturerSpecificDiagnosis together with DiagnosisData used as Status (no AlarmNotification) [c] The field ChannelProperties.Type, the field ChannelProperties.Direction, the field channelProperties.Accumulative shall be set to zero, the field ChannelProperties.Maintenance shall be set to Fault. The field ChannelProperties.Specifier shall be set to disappear.
ExtChannelDiagnosis	SlotNumber, SubslotNumber, ChannelNumber(0x8000), ChannelProperties [a], UserStructureIdentifier(0x8002), ExtChannelDiagnosisData* [a] The field ChannelProperties.Type, the field ChannelProperties.Direction, the field ChannelProperties.Maintenance shall be set to zero. The field ChannelProperties.Specifier shall be set to appear if at least one ChannelProperties.Specifier in the ExtChannelDiagnosisData is set to appear in conjunction with ChannelProperties.Maintenance(=Fault). Else, the field ChannelProperties.Specifier shall be set to disappear.
ExtChannelDiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType, ExtChannelErrorType, ExtChannelAddValue

Substitution name	Structure
QualifiedChannelDiagnosis	SlotNumber, SubslotNumber, ChannelNumber(0x8000), ChannelProperties [a], UserStructureIdentifier(0x8003), QualifiedChannelDiagnosisData* [a] The field ChannelProperties.Type, the field ChannelProperties.Direction, the field ChannelProperties.Maintenance shall be set to zero. The field ChannelProperties.Specifier shall be set to appear if at least one ChannelProperties.Specifier in the QualifiedChannelDiagnosisData is set to appear in conjunction with ChannelProperties.Maintenance(=Fault). Else, the field ChannelProperties.Specifier shall be set to disappear.
QualifiedChannel-DiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType, ExtChannelErrorType, ExtChannelAddValue, QualifiedChannelQualifier
ExpectedIdentificationData with BlockVersionLow =1	BlockHeader, NumberOfAPIs, (API, NumberOfSlots, (SlotNumber, ModuleIdentNumber, NumberOfSubslots, (SubslotNumber, SubmoduleIdentNumber)*)*)*
RealIdentificationData with BlockVersionLow = 1	BlockHeader, NumberOfAPIs, (API, NumberOfSlots, (SlotNumber, ModuleIdentNumber, NumberOfSubslots, (SubslotNumber, SubmoduleIdentNumber)*)*)*
PDIRData with BlockVersionLow = 1	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, PDIRGlobalData, PDIRFrameData [a] [b], PDIRBeginEndData [a] This block may contain information about receiving, transmitting, and forwarding, or receiving and transmitting only. [b] If this block exists, the DFP frames shall be integrated.
PDSyncData with BlockVersionLow = 2	BlockHeader, Padding, Padding, PTCP_SubdomainUUID, ReservedIntervalBegin, ReservedIntervalEnd, PLLWindow, SyncSendFactor, SendClockFactor, PTCPTimeoutFactor, PTCPTakeoverTimeoutFactor, PTCPMasterStartupTime, SyncProperties, PTCP_MasterPriority1, PTCP_MasterPriority2, PTCPLengthSubdomainName, PTCPSubdomainName, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDTimeData	BlockHeader, TimeMasterPriority1, TimeMasterPriority2, TimePLLWindow, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PdevData	BlockHeader, Padding, Padding, [PDIRData], [PDSyncData], [PDIRSubframeData]
PDRealData [c]	MultipleBlockHeader, { [PDPortDataReal] [b] [d], [PDPortDataRealExtended] [b] [d], [PDInterfaceMrpDataReal], [PDPortMrpDataReal], [PDPortFODataReal] [a], [PDInterfaceDataReal], [PDPortStatistic], [PDPortMrpIcDataReal] } [a] There shall be no FiberOpticManufacturerSpecific information [b] The fields SlotNumber and SubslotNumber shall be ignored [c] Each submodule's data (for example interface or port) needs its own MultipleBlockHeader [d] If MAUTypeExtension is used, both blocks shall be provided
PDExpectedData [b]	MultipleBlockHeader, { [PDPortDataCheck] [a], [PDPortDataAdjust] [a], [PDInterfaceMrpDataAdjust], [PDInterfaceMrpDataCheck], [PDPortMrpDataAdjust], [PDPortFODataAdjust], [PDPortFODataCheck], [PDNCDataCheck], [PDInterfaceFSUDataAdjust], [PDInterfaceAdjust], [PDPortMrpIcDataAdjust], [PDPortMrpIcDataCheck], [PDPortSFPDataCheck] } [a] The fields SlotNumber and SubslotNumber shall be ignored [b] Each submodule's data (for example interface or port) needs its own MultipleBlockHeader

Substitution name	Structure
PDPortDataReal	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, LengthOwnPortName, OwnPortName, NumberOfPeers, [Padding*] [a], [LengthPeerPortName, PeerPortName, LengthPeerStationName, PeerStationName, [Padding*] [a], LineDelay [e], PeerMACAddress [b], [Padding*] [a]]*, MAUType [c], [Padding*] [a], Reserved [d], RTClass3_PortStatus, MulticastBoundary, LinkState, [Padding*] [a], MediaType NOTE PDPort records are assigned to externally visible physical ports of end station or bridge components [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. [b] This field contains the interface MAC address of the peer [c] See Table 816 [d] The number of reserved octets shall be 2. [e] The local calculated LineDelay shall be used.
PDPortDataRealExtended	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, (OwnPort, [Neighbors] [a]) [a] This block shall be omitted if no peers exist
OwnPort	BlockHeader, Padding, Padding, LengthOwnPortName, OwnPortName, [Padding*] [a], LineDelay, MediaType, MulticastBoundary, MAUType, MAUTypeExtension, LinkState, RTClass3_PortStatus, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
Neighbors	BlockHeader, Padding, Padding, NumberOfPeers, [Padding*] [a], (LineDelay [c], MAUType [c], MAUTypeExtension [c], PeerMACAddress [b], LengthPeerPortName, PeerPortName, LengthPeerStationName, PeerStationName, [Padding*] [a])* [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. [b] This field contains the interface MAC address of the peer. [c] The values received by means of LLDP shall be used.
PDInterfaceDataReal	BlockHeader, LengthOwnStationName, OwnStationName, [Padding*] [a], MACAddressValue [b], [Padding*] [a], IPParameterValue, [Padding*] [a] NOTE PDInterface records are assigned to end station components. [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. [b] This field contains the interface MAC address
TxPortGroup	NumberOfTxPortGroups, TxPortGroupArray
SubstituteValue	BlockHeader, SubstitutionMode, SubstituteDataItem
RecordInputData-ObjectElement	BlockHeader, LengthIOCS, IOCS, LengthIOPS, IOPS, LengthData, Data Special case: Response for an output submodule PNIOStatus(=0xDE, 0x80, 0xB0, 0x00) Special case: Additional possible response for an input submodule without data PNIOStatus(=0xDE, 0x80, 0xB0, 0x00) or NULL [a] [a] NULL shall be used if a requested well-known data record is empty

Substitution name	Structure
RecordOutputData-ObjectElement	BlockHeader [c], SubstituteActiveFlag, LengthIOCS, LengthIOPS, LengthData, DataItem [a], SubstituteValue [b] Special case: Response for an input submodule PNIOStatus(=0xDE, 0x80, 0xB0, 0x00) [a] For this record DataItem shall be coded as IOCS, Data, IOPS [b] The IOPS of the SubstituteDataItem inherited by the SubstituteValue contains the valid information [c] The BlockLength shall include the SubstituteValue
ARData with BlockVersionLow = 0 Legacy Shall only be used with StartupMode:=Legacy	BlockHeader, NumberOfARs, (ARUUID, ARType, ARProperties, CMInitiatorObjectUUID, StationNameLength, CMInitiatorStationName, NumberOfIOCRs, (IOCRType, IOCRProperties, FrameID, APDU_Status [a], InitiatorUDPRTPort, ResponderUDPRTPort)*, AlarmCRType, LocalAlarmReference, RemoteAlarmReference, ParameterServerObjectUUID [b], StationNameLength [c], [ParameterServerStationName], NumberOfAPIs, API*)* Special case: “IOSAR with ARProperties.DeviceAccess=1”: NumberOfIOCRs := 0 AlarmCRType := 0 NumberOfAPIs := 0 [a] The APDU_Status.CycleCounter and APDU_Status.TransferStatus can be zero [b] The ParameterServerObjectUUID shall be NIL if not used [c] The StationNameLength shall be zero if not used. In this case the field ParameterServerStationName shall be omitted
ARData with BlockVersionLow = 1	BlockHeader, NumberOfARs, (ARUUID, CMInitiatorObjectUUID, ParameterServerObjectUUID [b], ARProperties, ARType, AlarmCRType, LocalAlarmReference, RemoteAlarmReference, InitiatorUDPRTPort, ResponderUDPRTPort, StationNameLength, CMInitiatorStationName, [Padding*] [d], StationNameLength [c], [ParameterServerStationName], [Padding*] [d], NumberOfIOCRs, [Padding*] [d], (IOCRProperties, IOCRType, FrameID, APDU_Status [a])*, NumberOfAPIs, [Padding*] [d], (API*), ARDataInfo)* Special case: “IOSAR with ARProperties.DeviceAccess=1”: NumberOfIOCRs := 0 AlarmCRType := 0 NumberOfAPIs := 0 [a] The APDU_Status.CycleCounter and APDU_Status.TransferStatus can be zero [b] The ParameterServerObjectUUID shall be NIL if not used [c] The StationNameLength shall be zero if not used. In this case the field ParameterServerStationName shall be omitted [d] The number of padding octets shall be adapted to make the following field Unsigned32 aligned.

Substitution name	Structure
ARData with BlockVersionLow = 2	BlockHeader, NumberOfARs, (ARUUID, CMInitiatorObjectUUID, ParameterServerObjectUUID [b], ARProperties, ARType, AlarmCRType, LocalAlarmReference, RemoteAlarmReference, InitiatorUDPRTPort, ResponderUDPRTPort, StationNameLength, CMInitiatorStationName, [Padding*] [d], StationNameLength [c], [ParameterServerStationName], [Padding*] [d], NumberOfIOCRs, [Padding*] [d], (IOCRProperties, IOCRType, FrameID, APDU_Status [a], SendClockFactor, ReductionRatio, Phase, Sequence, DataHoldFactor, [Padding*] [d])*, NumberOfAPIs, [Padding*] [d], (API*), ARDataInfo)* Special case: “IOSAR with ARProperties.DeviceAccess=1”: NumberOfIOCRs := 0 AlarmCRType := 0 NumberOfAPIs := 0 [a] The APDU_Status.CycleCounter and APDU_Status.TransferStatus can be zero [b] The ParameterServerObjectUUID shall be NIL if not used [c] The StationNameLength shall be zero if not used. In this case the field ParameterServerStationName shall be omitted [d] The number of padding octets shall be adapted to make the following field Unsigned32 aligned.
ARDataInfo	NumberOfEntries, [Padding*] [a], { [IRInfoBlock], [SRInfoBlock], [ARVendorBlockReq *], [ARVendorBlockRes *], [ARFSUBlock], [SRLData] [b], [RSInfoBlock], [ARAlgorithmInfoBlock] [c] } [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. [b] The SRLData block shall be used if a redundant network access point exists. [c] The ARAlgorithmInfoBlock block shall only be present if ARProperties.ProtectionProperties != 0.
SRLData	BlockHeader, RedundancyInfo, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
APIData	BlockHeader, NumberOfAPIs, API*
LogBookData with BlockVersionLow = 0	BlockHeader, ActualLocalTimeStamp, NumberOfLogEntries, (LocalTimeStamp, ARUUID, PNIOStatus, EntryDetail)*
LogBookData with BlockVersionLow = 1	BlockHeader, NumberOfLogEntries, Time_TimeStamp, (Time_TimeStamp, ARUUID, PNIOStatus, EntryDetail)*
CMInitiatorObjectUUID	PROFINETIOConstantValue, InstanceID, DeviceIdentNumber
ParameterServerObjectUUID	PROFINETIOConstantValue, InstanceID, DeviceIdentNumber
InstanceID	InstanceHigh, InstanceLow
DeviceID	DeviceIDHigh, DeviceIDLow
VendorID	VendorIDHigh, VendorIDLow
DeviceIdentNumber	DeviceID, VendorID
IMDataWrite	[I&M1] ^ [I&M2] ^ [I&M3] ^ [I&M4] [a] [a] Should only be used together with Functional Safety
IMData	I&M0 ^ [I&M1] ^ [I&M2] ^ [I&M3] ^ [I&M4] [a] ^ [I&M5] [a] Should only be used together with Functional Safety
I&M0	BlockHeader, VendorIDHigh, VendorIDLow, OrderID, IM_Serial_Number, IM_Hardware_Revision, IM_Software_Revision, IM_Revision_Counter, IM_Profile_ID, IM_Profile_Specific_Type, IM_Version, IM_Supported
I&M1	BlockHeader, IM_Tag_Function, IM_Tag_Location
I&M2	BlockHeader, IM_Date

Substitution name	Structure
I&M3	BlockHeader, IM_Descriptor
I&M4	BlockHeader, IM_Signature
I&M5	BlockHeader, NumberOfEntries, (I&M5Data ^ AssetManagementBlock [a])* [a] AM_Location shall be set to zero. Only reused for coding purposes – not part of Asset Management.
I&M5Data	BlockHeader, IM_Annotation, IM_OrderID, VendorIDHigh, VendorIDLow, IM_Serial_Number, IM_Hardware_Revision, IM_Software_Revision
I&M0FilterData	{I&M0FilterDataSubmodule [a], [I&M0FilterDataModule] [b], I&M0FilterDataDevice [c] } [a] Shall contain all submodules with discrete IMData. Supporting I&M0 is the precondition for the support of I&M1 to I&M5. [b] Shall, if it exists, contain only the module reference [c] Shall contain the device reference (for example the interface submodule) which shall support writing of I&M1, I&M2, and I&M3
I&M0FilterDataInfo	NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber)*)*)*
I&M0FilterDataSubmodule	BlockHeader, I&M0FilterDataInfo
I&M0FilterDataModule	BlockHeader, I&M0FilterDataInfo
I&M0FilterDataDevice	BlockHeader, I&M0FilterDataInfo
IM_Version	IM_Version_Major, IM_Version_Minor
IM_Software_Revision	SWRevisionPrefix, IM_SWRevision_Functional_Enhancement, IM_SWRevision_Bug_Fix, IM_SWRevision_Internal_Change
MultipleBlockHeader	BlockHeader, Padding, Padding, API, SlotNumber, SubslotNumber
PDIRGlobalData with BlockVersionLow = 1	BlockHeader, Padding, Padding, IRDataUUID, MaxBridgeDelay, NumberOfPorts, (MaxPortTxDelay, MaxPortRxDelay)*
PDIRGlobalData with BlockVersionLow = 2	BlockHeader, Padding, Padding, IRDataUUID, MaxBridgeDelay, NumberOfPorts, (MaxPortTxDelay, MaxPortRxDelay, MaxLineRxDelay, YellowTime)*
PDIRBeginEndData	BlockHeader, Padding, Padding, RedGuard, NumberOfPorts, (NumberOfAssignments, (TXBeginEndAssignment, RXBeginEndAssignment)*, NumberOfPhases, (TXPhaseAssignment, RXPhaseAssignment)*)*
RedGuard	StartOfRedFrameID, EndOfRedFrameID
TXBeginEndAssignment	RedOrangePeriodBegin, OrangePeriodBegin [a], GreenPeriodBegin [a] Shall be set to GreenPeriodBegin
RXBeginEndAssignment	RedOrangePeriodBegin, OrangePeriodBegin [a], GreenPeriodBegin [b] [a] Shall be set to GreenPeriodBegin [b] GreenPeriodBegin shall be set to the end of the red period.
TXPhaseAssignment	PhaseAssignment
RXPhaseAssignment	PhaseAssignment
PDIRFrameData with BlockVersionLow = 0	BlockHeader, Padding, Padding, (FrameSendOffset, DataLength, ReductionRatio, Phase, FrameID, EtherType, RxPort, FrameDetails, TxPortGroup, [Padding*] [a])* [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.

Substitution name	Structure
PDIRFrameData with BlockVersionLow = 1	BlockHeader, Padding, Padding, FrameDataProperties, [(FrameSendOffset, DataLength, ReductionRatio, Phase, FrameID [a] [b] [c], EtherType, RxPort, FrameDetails, TxPortGroup, [Padding*] [d])*] [a] This block shall contain all FrameIDs which are locally received (node is sink) or locally injected (node is source) into the network. It may contain the FrameIDs which are only forwarded by the node. [b] With DFP: This block shall contain two entries for the same FrameID if this node contains a DFP_RELAY_INBOUND or a DFP_RELAY_OUTBOUND state machine which is a receiver of this frame. One entry contains the receive path and one the transmit path. If the transmit path does not exist (outbound frame is fully consumed by this node), the corresponding entry shall be skipped. [c] With MRPD: A consumer shall exist twice (FrameID and FrameID+1) and a provider shall exist twice (FrameID and FrameID+1) in case of IOCR and may exist more than twice in case of MCR. [d] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
IsochronousModeData	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, ControllerApplicationCycleFactor, TimeDataCycle, TimeIOInput, TimeIOOutput, TimeIOInputValid, TimeIOOutputValid
PDPortDataAdjust	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, { [AdjustDomainBoundary], [AdjustMulticastBoundary], [AdjustMAUType ^ AdjustLinkState], [AdjustPeerToPeerBoundary], [AdjustDCPBoundary], [AdjustPreambleLength], [AdjustMAUTypeExtension] [a] } [a] Only possible if AdjustMAUType is part of the list
PDPortDataCheck	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, { [CheckPeers], [CheckLineDelay], [CheckMAUType [a]], [CheckLinkState], [CheckSyncDifference], [CheckMAUTypeDifference], [CheckMAUTypeExtension] [b] } [a] The implementation of the MAUType check shall take the AutoNegotiation settings into account. A diagnosis with severity fault shall be issued if an AutoNegotiation error is detected after the next LinkDown / LinkUp due to the used AutoNegotiation settings. [b] Only possible if CheckMAUType is part of the list
AdjustDomainBoundary with BlockVersionLow = 1	BlockHeader, Padding, Padding, DomainBoundaryIngress, DomainBoundaryEgress, AdjustProperties, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
AdjustMulticastBoundary	BlockHeader, Padding, Padding, MulticastBoundary, AdjustProperties, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
AdjustMAUType	BlockHeader, Padding, Padding, MAUType, AdjustProperties
AdjustMAUTypeExtension	BlockHeader, Padding, Padding, MAUTypeExtension, AdjustProperties
AdjustLinkState	BlockHeader, Padding, Padding, LinkState, AdjustProperties
AdjustPeerToPeerBoundary	BlockHeader, Padding, Padding, PeerToPeerBoundary, AdjustProperties, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
AdjustDCPBoundary	BlockHeader, Padding, Padding, DCPBoundary, AdjustProperties, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.

Substitution name	Structure
AdjustPreambleLength	BlockHeader, Padding, Padding, PreambleLength, AdjustProperties, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
CheckPeers	BlockHeader, NumberOfPeers, (LengthPeerPortName, PeerPortName, LengthPeerStationName, PeerStationName)*, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
CheckLineDelay	BlockHeader, Padding, Padding, LineDelay
CheckMAUType	BlockHeader, MAUType
CheckMAUTypeExtension	BlockHeader, MAUTypeExtension
CheckLinkState	BlockHeader, LinkState
CheckSyncDifference	BlockHeader, CheckSyncMode
CheckMAUTypeDifference	BlockHeader, MAUTypeMode
PDInterfaceAdjust	BlockHeader, Padding, Padding, MultipleInterfaceMode, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDInterfaceMrpDataAdjust with BlockVersionLow = 0	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], [(MrpManagerParams) ^ (MrpClientParams)] Special case MRP_Role := “Media Redundancy disabled”: BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a] Special case MRP_Role := “Media Redundancy Client”: BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], MrpClientParams Special case MRP_Role := “Media Redundancy Manager”: BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], MrpManagerParams [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDInterfaceMrpDataAdjust with BlockVersionLow = 1	BlockHeader, Padding, MRP_NumberOfEntries, MrpInstanceDataAdjustBlock*
PDInterfaceMrpDataReal with BlockVersionLow = 1	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, MRP_Version, MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], { [(MrpManagerParams) ^ (MrpClientParams)], [MrpRingStateData] }, [Padding*] [a] Special case MRP_Role := “Media Redundancy disabled”: BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, MRP_Version, MRP_LengthDomainName, MRP_DomainName, [Padding*] [a] Special case MRP_Role := “Media Redundancy Client”: BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, MRP_Version, MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], MrpClientParams, [Padding*] [a] Special case MRP_Role := “Media Redundancy Manager”: BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, MRP_Version, MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], { MrpManagerParams, MrpRingStateData }, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.

Substitution name	Structure
PDInterfaceMrpDataReal with BlockVersionLow = 2	BlockHeader, Padding, MRP_NumberOfEntries, MrpInstanceDataRealBlock*
PDInterfaceMrpDataCheck with BlockVersionLow = 0	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Check
PDInterfaceMrpDataCheck with BlockVersionLow = 1	BlockHeader, Padding, MRP_NumberOfEntries, MrpInstanceDataCheckBlock*
PDPortMrpDataAdjust with BlockVersionLow = 0	BlockHeader, Padding, Padding, MRP_DomainUUID
PDPortMrpDataAdjust with BlockVersionLow = 1	BlockHeader, Padding, MRP_Instance, MRP_DomainUUID
PDPortMrpDataReal with BlockVersionLow = 0	BlockHeader, Padding, Padding, MRP_DomainUUID
PDPortMrpDataReal with BlockVersionLow = 1	BlockHeader, Padding, MRP_Instance, MRP_DomainUUID
MrpManagerParams	BlockHeader, MRP_Prio, MRP_TOPchgT, MRP_TOPNRmax, MRP_TSTshortT, MRP_TSTdefaultT, MRP_TSTNRmax, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
MrpClientParams	BlockHeader, MRP_LNKdownT, MRP_LNKupT, MRP_LNKNRmax
MrpRingStateData	BlockHeader, MRP_RingState
MrpInstanceDataAdjustBlock	BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], { [MrpManagerParams], [MrpClientParams] } Special case MRP_Role := “Media Redundancy disabled”: BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a] Special case MRP_Role := “Media Redundancy Client”: BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], [MrpClientParams] Special case MRP_Role := “Media Redundancy Manager”: BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], MrpManagerParams Special case MRP_Role := “Media Redundancy Manager (Auto manager negotiation)”: BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], { MrpManagerParams, [MrpClientParams] } [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.

Substitution name	Structure
MrpInstanceDataRealBlock	BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Role, MRP_Version, MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], { [(MrpManagerParams) ^ (MrpClientParams)], [MrpRingStateData] }, [Padding*] [a] Special case MRP_Role := “Media Redundancy disabled”: BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Role, MRP_Version, MRP_LengthDomainName, MRP_DomainName, [Padding*] [a] Special case MRP_Role := “Media Redundancy Client”: BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Role, MRP_Version, MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], MrpClientParams, [Padding*] [a] Special case MRP_Role := “Media Redundancy Manager” or MRP_Role := “Media Redundancy Manager (Auto manager negotiation)”: BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Role, MRP_Version, MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], { MrpManagerParams, MrpRingStateData }, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
MrpInstanceDataCheckBlock	BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Check
PDPortFODataReal	BlockHeader, Padding, Padding, FiberOpticType, FiberOpticCableType, [FiberOpticManufacturerSpecific*] [a], [FiberOpticDiagnosisInfo] [a] Only used by legacy nodes.
FiberOpticManufacturerSpecific	BlockHeader, VendorIDHigh, VendorIDLow, VendorBlockType, Data*, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
FiberOpticDiagnosisInfo	BlockHeader, Padding, Padding, FiberOpticPowerBudgetReal, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
FiberOpticPowerBudgetReal	FiberOpticPowerBudgetType [a] [a] FiberOpticPowerBudgetType.CheckEnable shall be set to zero. It has no meaning in this usage.
PDPortFODataAdjust	BlockHeader, Padding, Padding, FiberOpticType, FiberOpticCableType
PDPortFODataCheck	BlockHeader, Padding, Padding, MaintenanceRequiredPowerBudget, MaintenanceDemandedPowerBudget, ErrorPowerBudget
MaintenanceRequired-PowerBudget	FiberOpticPowerBudgetType
MaintenanceDemanded-PowerBudget	FiberOpticPowerBudgetType
ErrorPowerBudget	FiberOpticPowerBudgetType
PDPortSFPDataCheck	BlockHeader, Padding, Padding, MaintenanceDemandedAdminStatus, ErrorAdminStatus
PDNCDataCheck	BlockHeader, Padding, Padding, MaintenanceRequiredDropBudget, MaintenanceDemandedDropBudget, ErrorDropBudget
MaintenanceRequired-DropBudget	NCDropBudgetType
MaintenanceDemanded-DropBudget	NCDropBudgetType
ErrorDropBudget	NCDropBudgetType
PDPortStatistic with BlockVersionLow = 0	BlockHeader, Padding, Padding, ifInOctets, ifOutOctets, ifInDiscards, ifOutDiscards, ifInErrors, ifOutErrors, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.

Substitution name	Structure
PDPortStatistic with BlockVersionLow = 1	BlockHeader, CounterStatus, ifInOctets, ifOutOctets, ifInDiscards, ifOutDiscards, ifInErrors, ifOutErrors, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDInterfaceFSUDataAdjust	BlockHeader, Padding, Padding, { [FSHelloBlock], [FastStartUpBlock] } [a] [a] At least one optional block shall be existing.
ARFSUDataAdjust	BlockHeader, Padding, Padding, { [FSParameterBlock], [FastStartUpBlock] } [a] [a] At least one optional block shall be existing.
FSHelloBlock	BlockHeader, Padding, Padding, FSHelloMode, FSHelloInterval, FSHelloRetry, FSHelloDelay
FSParameterBlock	BlockHeader, Padding, Padding, FSParameterMode, FSParameterUUID
FastStartUpBlock	BlockHeader, Padding, Padding, Data*, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
CombinedObjectContainer	(COContainerContent, [Padding]* [a])* [a] The number of padding octets (value=0) shall be 0,1,2,3 to have 32 bit alignment to the next COContainerContent
COContainerContent	BlockHeader, Padding, Padding, API, Slot, Subslot, Padding, Padding, Index, COContainerBlock
COContainerBlock	RecordDataWrite [a] [a] Without CombinedObjectContainer
UploadBLOBQuery	BlockHeader, Padding, Padding, FromOffsetData
UploadBLOB	BlockHeader, Padding, Padding, FromOffsetData, NextOffsetData, TotalSize, Data*
PE_EntityFilterData	BlockHeader, PE_FilterDataInfo
PE_FilterDataInfo	NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber)*)*)*
PE_EntityStatusData	BlockHeader, PE_StatusDataInfo
PE_StatusDataInfo	NumberOfAPIs, (API, NumberOfModules, (SlotNumber, NumberOfSubmodules, (SubslotNumber, PE_OperationalMode, [Padding]* [a])*)*)* [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
AssetManagementData	BlockHeader, AssetManagementInfo
AssetManagementInfo	NumberOfEntries, AssetManagementBlock*
AssetManagementBlock	BlockHeader, Padding, Padding, AM_Info
AM_Info	AM_FullInformation ^ AM_FirmwareOnlyInformation ^ AM_HardwareOnlyInformation
AM_FullInformation	IM_UniqueIdentifier, AM_Location, IM_Annotation, IM_OrderID, AM_SoftwareRevision, AM_HardwareRevision, IM_Serial_Number, IM_Software_Revision, AM_DeviceIdentification, AM_TypeIdentification, IM_Hardware_Revision
AM_HardwareOnlyInformation	IM_UniqueIdentifier, AM_Location, IM_Annotation, IM_OrderID, AM_HardwareRevision, IM_Serial_Number, AM_DeviceIdentification, AM_TypeIdentification, IM_Hardware_Revision
AM_FirmwareOnlyInformation	IM_UniqueIdentifier, AM_Location, IM_Annotation, IM_OrderID, AM_SoftwareRevision, IM_Serial_Number, IM_Software_Revision, AM_DeviceIdentification, AM_TypeIdentification, AM_Reserved
RS_AdjustObserver	BlockHeader, RS_AdjustInfo
RS_AdjustInfo	NumberOfEntries, RS_AdjustBlock*

Substitution name	Structure
RS_AdjustBlock	RS_BlockHeader, [Padding]* [a], RS_AdjustControl, [Padding]* [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
RS_BlockHeader	RS_BlockType, RS_BlockLength, BlockVersionHigh, BlockVersionLow
RS_AdjustControl	RSO_DigitalInputObserver ^ Data*
RSO_DigitalInputObserver	ChannelNumber, RS_MaxScanDelay, RS_AdjustSpecifier
RS_GetEvent	BlockHeader, RS_EventInfo
RS_EventInfo	NumberOfEntries, RS_EventBlock*
RS_EventBlock	RS_BlockHeader, [Padding]* [a], RS_EventData
RS_EventData	RS_EventDataCommon, [RS_EventDataExtension]*, [Padding]* [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
RS_EventDataCommon	RS_AddressInfo, RS_Specifier, Time_TimeStamp, RS_MinusError, RS_PlusError
RS_AddressInfo	IM_UniqueIdentifier, API, SlotNumber, SubslotNumber, ChannelNumber Special case “SlotNumber := 0x8000”: SubslotNumber shall be set to zero and ChannelNumber shall be set to 0x8000.
RS_EventDataExtension	RS_ExtensionBlockType, RS_ExtensionBlockLength, Data*, [Padding]* [a] Special case “RSO_BufferObserver”: — Special case “RSO_StatusObserver”: RS_ExtensionBlockType(=0x80), RS_ExtensionBlockLength, RS_BlockType [a], RS_ReasonCode Special case “RSO_TimeStatus”: RS_ExtensionBlockType(=0x80), RS_ExtensionBlockLength, [Padding]* [a], RS_DomainIdentification, RS_MasterIdentification, [Time_TimeStamp] [a] Special case “RSO_DigitalInputObserver”: RS_ExtensionBlockType(=0x80), RS_ExtensionBlockLength, RS_DigitalInputCurrentValue Special case “RSO_SRLObserver”: — Special case “RSO_SourceIdentification”: RS_ExtensionBlockType(=0x80), RS_ExtensionBlockLength, RS_IdentificationInfo [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. [b] Contains the type of the reported observer. [c] Shall be used if a leap in time takes place.
RS_IdentificationInfo	AM_DeviceIdentification, IM_Tag_Function, IM_Tag_Location
RS_AckEvent	BlockHeader, RS_AckInfo
RS_AckInfo	RS_Specifier

Substitution name	Structure
PDPortMrpIcDataAdjust	BlockHeader, Padding, Padding, MRPIC_DomainID, MRPIC_Role, MRPIC_LengthDomainName, MRPIC_DomainName, [Padding] [a], MRPIC_MIMParams ^ MRPIC_MICParams Special case “MRPIC_Role := MIM”: BlockHeader, Padding, Padding, MRPIC_DomainID, MRPIC_Role, MRPIC_LengthDomainName, MRPIC_DomainName, [Padding] [a], [MRPIC_MIMParams] Special case “MRPIC_Role := MIC”: BlockHeader, Padding, Padding, MRPIC_DomainID, MRPIC_Role, MRPIC_LengthDomainName, MRPIC_DomainName, [Padding] [a], [MRPIC_MICParams] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDPortMrpIcDataCheck	BlockHeader, Padding, Padding, MRPIC_Check
PDPortMrpIcDataReal	BlockHeader, Padding, Padding, MRPIC_DomainID, MRPIC_Role, MRP_Version, MRPIC_LengthDomainName, MRPIC_DomainName, [Padding] [a], (MRPIC_MIMParams, MRPIC_StateData) ^ MRPIC_MICParams Special case “MRPIC_Role := MIM”: BlockHeader, Padding, Padding, MRPIC_DomainID, MRPIC_Role, MRP_Version, MRPIC_LengthDomainName, MRPIC_DomainName, [Padding] [a], MRPIC_MIMParams, MRPIC_StateData Special case “MRPIC_Role := MIC”: BlockHeader, Padding, Padding, MRPIC_DomainID, MRPIC_Role, MRP_Version, MRPIC_LengthDomainName, MRPIC_DomainName, [Padding] [a], MRPIC_MICParams [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
MRPIC_MIMParams	MRPIC_TOPchgT, MRPIC_TOPNRmax, MRPIC_LinkStatusChangeT, MRPIC_LinkStatusNRmax, MRPIC_StartDelay
MRPIC_MICParams	MRPIC_LNKdownT, MRPIC_LNKupT, MRPIC_LNKNRmax, MRPIC_StartDelay, MRPIC_MICPosition, Padding, Padding
MRPIC_StateData	MRPIC_State
SystemIdentification	DeviceType, Blank, OrderID, Blank, IM_Serial_Number, Blank, HWRevision, Blank, SWRevisionPrefix, SWRevision
ExtendedSystemIdentification	OctetString[64], SystemIdentification
ApplicationReadyBlock	[ModuleDiffBlock, [DiagnosisData* [a]]] [a] DiagnosisData with BlockVersionLow=1 and the format of index 0xE00C shall be used.
PDRsiInstances	BlockHeader, NumberOfEntries, (VendorID, DeviceID, InstanceID, RsiInterface, Padding)*, SystemIdentification
CIMElectricPowerReal	BlockHeader, NumberOfEntries, [ElectricPowerPortData *], ElectricPowerDeviceData [a] [a] Shall be provided if MAUTypeExtension := APL is supported
ElectricPowerDeviceData	CIMElementID, ElectricPowerDeviceVoltage
ElectricPowerPortData	CIMElementID, ElectricPowerPortVoltage, ElectricPowerPortCurrent

Substitution name	Structure
CIMSNMPAdjust	BlockHeader, SNMPControl, SNMPReadOnlyCommunityName, SNMPReadWriteCommunityName, [Padding] [c] Special case: SNMPControl.SNMPControl := Disable SNMP BlockHeader, SNMPControl, SNMPReadOnlyCommunityName [a], SNMPReadWriteCommunityName [a], [Padding] [c] Special case: SNMPControl.SNMPControl := Enable SNMP, Disable Write BlockHeader, SNMPControl, SNMPReadOnlyCommunityName [b], SNMPReadWriteCommunityName [a], [Padding] [c] Special case: SNMPControl.SNMPControl := Enable SNMP, Enable Read and Write BlockHeader, SNMPControl, SNMPReadOnlyCommunityName [b], SNMPReadWriteCommunityName [b], [Padding] [c] [a] The CommunityNameLength shall be set to zero. [b] The field CommunityNameLength controls the writing of a community name. The value zero means no writing. [c] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
SNMPCommunityName	CommunityNameLength [a], [CommunityName] [a] When used within any read service, the value shall be zero.
SNMPReadOnly-CommunityName	SNMPCommunityName
SNMPReadWrite-CommunityName	SNMPCommunityName

Table 539 – IO APDU substitutions for CIM

Substitution name	Structure
CIMDataRead	CIMNetConfUploadNetworkAttributes ^ CIMNetConfDataReal ^ CIMNetConfDataAdjust ^ CIMNetConfStreamPathDataReal ^ CIMNetConfSyncTreeData ^ CIMNetConfExpectedNetworkAttributes ^ CIMSNMPAdjust ^ CIMElectricPowerReal
CIMDataWrite	CIMNetConfDataAdjust ^ CIMNetConfStreamPathData ^ CIMNetConfSyncTreeData ^ CIMNetConfExpectedNetworkAttributes ^ CIMSNMPAdjust
CIMNetConfDataAdjust	BlockHeader, Padding, Padding, NMEParameterUUID, NMEDomainVIDConfig, CIMStationPortConfigBlock, NetworkDeadline, GatingCycle, NumberOfEntries, [NMEDomainTimeDataBlock*], NMENameUUID, NMENameAddress, NMENameLength, NMEName, [Padding*] [a], [TrafficClassTranslationTable], [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
CIMNetConfDataReal	BlockHeader, Padding, Padding, NMEParameterUUID, NMEDomainVIDConfig, CIMStationPortConfigBlock, NetworkDeadline, GatingCycle, NumberOfEntries, [NMEDomainTimeDataBlock*], NMENameUUID, NMENameAddress, NMENameLength, NMEName, [Padding*] [a], NMEDomainUUID, NMEDomainNameLength, NMEDomainName, [Padding*] [a], NMEDomainConfigRealBlock, [TrafficClassTranslationTable] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
CIMStation-PortConfigBlock	BlockHeader, NumberOfEntries, (CIMStationElementID [a], CIMStationPortStatus [b], Padding, Padding, Padding, CIMStationPortIngressRateLimiterBlock, CIMStationQueueConfigBlock, CIMStationEgressRateLimiterBlock) * [a] Either SubslotNumber 0x8000 or 0xA020 in combination with 0xA040 can be used. If SubslotNumber 0x8000 is used, it is a synonym for 0xA020 and 0xA040. In this case apply for the element addressed by 0xA040 the CIMStationEgressRateLimiterBlock, and for the element addressed by 0xA020 all other settings. [b] CIMStationPortStatus shall be set to zero on the write service and ignored by the receiver.

Substitution name	Structure
CIMStationPortIngress-RateLimiterBlock	BlockHeader, NumberOfEntries [a], [BroadcastPortIngressRateLimiter, MulticastPortIngressRateLimiter, UnicastPortIngressRateLimiter, PortIngressRateLimiter [b], PortIngressRateLimiter [c]] [a] Zero, three or five entries [b] RT_CLASS_X [c] RTA_CLASS_X
BroadcastPortIngress-RateLimiter	PortIngressRateLimiter
MulticastPortIngress-RateLimiter	PortIngressRateLimiter
UnicastPortIngress-RateLimiter	PortIngressRateLimiter
CIMStation-QueueConfigBlock	BlockHeader, NumberOfEntries, [NMEDomainQueueConfig*]
CIMStation-EgressRateLimiterBlock	BlockHeader, NumberOfEntries [a], [PortQueueEgressRateLimiter*] [a] Zero or five entries.
NMEDomain-TimeDataBlock	BlockHeader, TimeDomainNumber, TimePLLWindow, MessageIntervalFactor, MessageTimeoutFactor, TimeSyncProperties, TimeDomainUUID, TimeDomainNameLength, TimeDomainName, [Padding *] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
CIMStationElementID	SlotNumber, SubslotNumber [a] [a] The SubslotNumber shall be set according Table 556, Formula (53) and Table 616.
IngressPort	CIMStationElementID
EgressPort	CIMStationElementID
CIMNetConf-StreamPathData	BlockHeader, Padding, FDBCommand, Padding, Padding, NumberOfEntries, (DestinationAddress, StreamClass, IngressPort [a], EgressPort [b]) *, CIMStreamCollectionUUID Special case "FDBCommand == RemoveAllStreamEntries": BlockHeader, Padding, FDBCommand, Padding, Padding, NumberOfEntries(0), CIMStreamCollectionUUID [a] This is only used for information. [b] The bridge component is addressed according to the EgressPort.
CIMNetConf-StreamPathDataReal	BlockHeader, Padding, Padding, Padding, Padding, NumberOfEntries, (DestinationAddress, StreamClass, IngressPort [a], EgressPort) * [a] This is only used for information.
CIMNetConf-SyncTreeData	BlockHeader, NumberOfEntries, (CIMStationElementID, TimeDomainNumber, SyncPortRole, Padding) *, CIMSyncTreeDataUUID
CIMNetConfUpload-NetworkAttributes	BlockHeader, Padding, Padding, CIMStationPortCapabilitiesBlock, MaxSupportedRecordSize, TransferTimeTX, TransferTimeRX, CIMStationForwardingDelayBlock, SupportedBurstSize, MinIPGBreakingPoint, MinIPGFrameSize, FrameSendOffsetDeviation
CIMStation-PortCapabilitiesBlock	BlockHeader, NumberOfEntries, (CIMStationElementID [a], MAUType, MAUTypeExtension, NumberOfQueues, PortCapabilities, ForwardingGroup, Padding) * [a] Either SubslotNumber 0x8000 or 0xA020 in combination with 0xA040 can be used. If SubslotNumber 0x8000 is used, it is a synonym for 0xA020. The chosen SubslotNumber coding reported via this block shall be used together with the CIMStationPortConfigBlock, IngressPort, EgressPort and CIMNetConfSyncTreeData.
CIMStation-ForwardingDelayBlock	BlockHeader, NumberOfEntries, (ForwardingGroupIngress, ForwardingGroupEgress, StreamClass [a], ForwardingDelay) * [a] Only StreamClass High and StreamClass Low are supported.

Substitution name	Structure
ForwardingGroup-Ingress	ForwardingGroup
ForwardingGroupEgress	ForwardingGroup
CIMNetConfExpected-NetworkAttributes	BlockHeader, Padding, Padding, CIMStationPortCapabilitiesBlock, CIMStationForwardingDelayBlock, CIMStationExpectedNeighborBlock ,CIMExpectedNetworkAttributesUUID [a] The field ForwardingGroup shall be ignore for the write service.
CIMStation-ExpectedNeighborBlock	BlockHeader, NumberOfEntries, (CIMStationElementID, Padding, Padding, Padding, NumberOfPeers(1), [CIMStationExpectedNeighbor*])*
CIMStation-ExpectedNeighbor	LengthPeerPortName, PeerPortName, LengthPeerStationName, PeerStationName, [Padding*] [a], LineDelay [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
NMEDomain-ConfigRealBlock	BlockHeader, Padding, Padding, CIMStreamCollectionUUID, CIMSyncTreeDataUUID, CIMExpectedNetworkAttributesUUID
NMENameAddress	NMENameAddressSubtype, (Padding, Padding, Padding, IPAddress) ^ (Padding, MACAddressValue)
TrafficClass-TranslationTable	TrafficClassTranslateEntry[10]

Table 540 – IO APDU substitutions for UNI

Substitution name	Structure
UNIAddStreamReq	BlockHeader, NumberOfEntries, (StreamIdentification, StreamSource, StreamSink, StreamControl, PduSize, MinUpdateInterval, MaxUpdateInterval, ApplicationDeadline)*
StreamSource	StreamEndpoint
StreamSink	StreamEndpoint
StreamEndpoint	NMEDomainUUID, StationNameLength [a], [EndpointStationName], [Padding*] [b] [a] The StationNameLength shall be zero if not used. In this case the field EndpointStationName shall be omitted. [b] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
MinUpdateInterval	UpdateInterval
MaxUpdateInterval	Padding, Padding
UNIAddStreamRsp	BlockHeader, NumberOfEntries, (PNIOStatus, StreamIdentification, SendClockFactor, ReductionRatio, Phase, Sequence, FrameSendOffset [a], StreamTCI, DestinationAddress, MaxCalculatedLatency)* [a] Shall set to zero. Not used for this service.
UNIRemoveStreamReq	BlockHeader, Padding, Padding, StreamType, NumberOfEntries, (StreamIdentification, [StreamEndpoint])* Special case “StreamType := Unicast”: BlockHeader, Padding, Padding, StreamType, NumberOfEntries, (StreamIdentification)* Special case “StreamType := Multicast”: BlockHeader, Padding, Padding, StreamType, NumberOfEntries, (StreamIdentification, StreamEndpoint)*

Substitution name	Structure
UNIRemoveStreamRsp	BlockHeader, Padding, Padding, StreamType, NumberOfEntries, (StreamIdentification, [StreamEndpoint], PNIOStatus)* Special case “StreamType := Unicast”: BlockHeader, Padding, Padding, StreamType, NumberOfEntries, (StreamIdentification, PNIOStatus)* Special case “StreamType := Multicast”: BlockHeader, Padding, Padding, StreamType, NumberOfEntries, (StreamIdentification, StreamEndpoint, PNIOStatus)*
UNIRenewStreamReq	BlockHeader, NumberOfEntries, (StreamIdentification)*
UNIRenewStreamRsp	BlockHeader, NumberOfEntries, (StreamIdentification, PNIOStatus)*

Table 541 – IO APDU substitutions for security

Substitution name	Structure
IOXSecureReq	Reserved for security
IOXSecureRsp	Reserved for security
CIMSecurityServiceReq	Reserved for security
CIMSecurityServiceRsp	Reserved for security

Table 542 – IO APDU substitutions for CIM services

Substitution name	Structure
CIMDCPServiceReq	CIMDCPGetBlockReq ^ CIMDCPSetBlockReq
CIMDCPServiceRsp	CIMDCPGetBlockRsp ^ CIMDCPSetBlockRsp
CIMStationComponentID	SlotNumber [a], SubslotNumber [b] NOTE Used to identify one of the possible 16 end station components/interfaces. See “I” within Formula (53). [a] Set to zero by the requester and ignored by the responder. [b] The SubslotNumber shall be set according Table 556, Formula (53) and Table 616.
CIMDCPGetBlockReq	BlockHeader, Padding, Padding, CIMStationComponentID, GetReqBlock*
CIMDCPGetBlockRsp	BlockHeader, Padding, Padding, CIMStationComponentID, (GetResBlock ^ GetNegResBlock)*
CIMDCPSetBlockReq	BlockHeader, Padding, Padding, CIMStationComponentID, SetResetReqBlock ^ SetReqBlock*
CIMDCPSetBlockRsp	BlockHeader, Padding, Padding, CIMStationComponentID, (SetResBlock ^ SetNegResBlock)*

5.2 Transfer syntax

5.2.1 Coding section related to BlockHeader specific fields

5.2.1.1 Coding of the field BlockType

This field shall be coded as data type Unsigned16 with the values according to Table 543. This field identifies the PDU or data block type.

NOTE BlockType is an administrative number and thus unique in the context of this document, except for the vendor specific portion.

Table 543 – BlockType

Value (hexadecimal)	Used for	Used by
0x0000	Reserved	—
0x0001	AlarmNotification High	RTA-SDU.AlarmNotification-PDU
0x0002	AlarmNotification Low	RTA-SDU.AlarmNotification-PDU
0x0003 – 0x0007	Reserved	—
0x0008	IODWriteReqHeader	PROFINETIO-ServiceReqPDU.IODWriteReq
0x0009	IODReadReqHeader	PROFINETIO-ServiceReqPDU.IODReadReq
0x000A – 0x000F	Reserved	—
0x0010	DiagnosisData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.DiagnosisData RTA-SDU.AlarmNotification-PDU.AlarmPayload.DiagnosisItem.DiagnosisData
0x0011	Reserved	—
0x0012	ExpectedIdentificationData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.ExpectedIdentificationData
0x0013	RealIdentificationData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.RealIdentificationData
0x0014	SubstituteValue	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.SubstituteValue PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.SubstituteValue
0x0015	RecordInputDataObjectElement	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.RecordInputDataObjectElement
0x0016	RecordOutputDataObjectElement	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.RecordOutputDataObjectElement
0x0017	Reserved	—
0x0018	ARData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.ARData
0x0019	LogBookData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.LogBookData
0x001A	APIData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.APIData
0x001B	SRLData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.ARDataInfo.SRLData
0x001C – 0x001F	Reserved	—
0x0020	I&M0	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.IMData.I&M0

Value (hexadecimal)	Used for	Used by
0x0021	I&M1	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.IMData.I&M1 PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.IMDataWrite.I&M1
0x0022	I&M2	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.IMData.I&M2 PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.IMDataWrite.I&M2
0x0023	I&M3	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.IMData.I&M3 PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.IMDataWrite.I&M3
0x0024	I&M4	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.IMData.I&M4 PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.IMDataWrite.I&M4
0x0025	I&M5	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.IMData.I&M5
0x0026 – 0x002F	I&M6 – I&M15 [a] [a] Reserved for additional identification and maintenance data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite
0x0030	I&M0FilterDataSubmodule	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.I&M0FilterData.I&M0FilterDataSubmodule
0x0031	I&M0FilterDataModule	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.I&M0FilterData.I&M0FilterDataModule
0x0032	I&M0FilterDataDevice	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.I&M0FilterData.I&M0FilterDataDevice
0x0033	Reserved	—
0x0034	I&M5Data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.IMData.I&M5.I&M5Data
0x0035	AssetManagementData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.AssetManagementData
0x0036	Asset Management – Full information	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.AssetManagementData.AM_FullInformation PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.I&M5.AM_FullInformation

Value (hexadecimal)	Used for	Used by
0x0037	Asset Management – Only hardware information	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.AssetManagementData.AM_HardwareOnlyInformation PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.I&M5.AM_HardwareOnlyInformation
0x0038	Asset Management – Only firmware information	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.AssetManagementData.AM_FirmwareOnlyInformation PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.I&M5.AM_FirmwareOnlyInformation
0x0039 – 0x003F	Reserved for Asset Management	—
0x0040 – 0x0100	Reserved	—
0x0101	ARBlockReq	PROFINETIO-ServiceReqPDU.IODConnectReq.ARBlockReq
0x0102	IOCRBlockReq	PROFINETIO-ServiceReqPDU.IODConnectReq.IOCRBlockReq
0x0103	AlarmCRBlockReq	PROFINETIO-ServiceReqPDU.IODConnectReq.AlarmCRBlockReq
0x0104	ExpectedSubmoduleBlockReq	PROFINETIO-ServiceReqPDU.IODConnectReq.ExpectedSubmoduleBlockReq
0x0105	PrmServerBlock	PROFINETIO-ServiceReqPDU.IODConnectReq.PrmServerBlock
0x0106	MCRBlockReq	PROFINETIO-ServiceReqPDU.IODConnectReq.MCRBlockReq
0x0107	ARRPCBlockReq	PROFINETIO-ServiceReqPDU.IODConnectReq.ARRPCBlockReq
0x0108	ARVendorBlockReq	PROFINETIO-ServiceReqPDU.IODConnectReq.ARVendorBlockReq
0x0109	IRInfoBlock	PROFINETIO-ServiceReqPDU.IODConnectReq.IRInfoBlock
0x010A	SRInfoBlock	PROFINETIO-ServiceReqPDU.IODConnectReq.SRInfoBlock
0x010B	ARFSUBlock	PROFINETIO-ServiceReqPDU.IODConnectReq.ARFSUBlock
0x010C	RSInfoBlock	PROFINETIO-ServiceReqPDU.IODConnectReq.RSInfoBlock PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.ARDataInfo.RSInfoBlock
0x010D	ARAlgorithmInfoBlock	PROFINETIO-ServiceReqPDU.IODConnectReq.ARAlgorithmInfoBlock PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.ARDataInfo.ARAlgorithmInfoBlock
0x010E – 0x010F	Reserved	—
0x0110	IODBlockReq, shall only be used in conjunction with connection establishment phase or a PrmBegin/PrmEnd sequence	PROFINETIO-ServiceReqPDU.IODControlReq.ControlBlockConnect (PrmEnd.req)
0x0111	IODBlockReq, shall only be used in conjunction with a plug alarm event	PROFINETIO-ServiceReqPDU.IODControlReq.ControlBlockPlug (PrmEnd.req)

Value (hexadecimal)	Used for	Used by
0x0112	IOXBlockReq, shall be used in conjunction with the connection establishment phase or a PrmBegin/PrmEnd sequence	PROFINETIO-ServiceReqPDU.IOXControlReq.ControlBlockConnect (ApplicationReady.req)
0x0113	IOXBlockReq, shall only be used in conjunction with a plug alarm event	PROFINETIO-ServiceReqPDU.IOXControlReq.ControlBlockPlug (ApplicationReady.req)
0x0114	ReleaseBlockReq	PROFINETIO-ServiceReqPDU.IODReleaseReq.ReleaseBlock
0x0115	Reserved	—
0x0116	IOXBlockReq, shall only be used in conjunction with connection establishment phase	PROFINETIO-ServiceReqPDU.IOXControlReq.ControlBlockConnect (ReadyForCompanion.req)
0x0117	IOXBlockReq, shall only be used in conjunction with connection establishment phase	PROFINETIO-ServiceReqPDU.IOXControlReq.ControlBlockConnect (ReadyForRT_CLASS_3.req)
0x0118	IODBlockReq	PROFINETIO-ServiceReqPDU.IODControlReq.ControlBlockConnect (PrmBegin.req)
0x0119	SubmoduleListBlock	PROFINETIO-ServiceReqPDU.IODControlReq.SubmoduleListBlock (PrmBegin.req)
0x011A	SecurityRequestBlock	Reserved for security
0x011B – 0x01FF	Reserved	—
0x0200	PDPortDataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDPortDataCheck
0x0201	PdevData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PdevData
0x0202	PDPortDataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDPortDataAdjust
0x0203	PDSyncData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDSyncData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDSyncData
0x0204	IsochronousModeData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.IsochronousModeData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.IsochronousModeData

Value (hexadecimal)	Used for	Used by
0x0205	PDIRData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDIRData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDIRData
0x0206	PDIRGlobalData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDIRGlobalData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDIRGlobalData
0x0207	PDIRFrameData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDIRFrameData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDIRFrameData
0x0208	PDIRBeginEndData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDIRBeginEndData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDIRBeginEndData
0x0209	AdjustDomainBoundary Sub block for adjusting DomainBoundary	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataAdjust.AdjustDomainBoundary PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust.AdjustDomainBoundary
0x020A	CheckPeers Sub block for checking Peers	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataCheck.CheckPeers PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataCheck.CheckPeers
0x020B	CheckLineDelay Sub block for checking LineDelay	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataCheck.CheckLineDelay PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataCheck.CheckLineDelay
0x020C	CheckMAUType Sub block for checking MAUType	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataCheck.CheckMAUType PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataCheck.CheckMAUType
0x020E	AdjustMAUType Sub block for adjusting MAUType	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataAdjust.AdjustMAUType PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust.AdjustMAUType

Value (hexadecimal)	Used for	Used by
0x020F	PDPortDataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDPortDataReal
0x0210	AdjustMulticastBoundary Sub block for adjusting MulticastBoundary	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataAdjust.AdjustMulticastBoundary PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust.AdjustMulticastBoundary
0x0211	PDInterfaceMrpDataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceMrpDataAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDInterfaceMrpDataAdjust
0x0212	PDInterfaceMrpDataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDInterfaceMrpDataReal
0x0213	PDInterfaceMrpDataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceMrpDataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDInterfaceMrpDataCheck
0x0214	PDPortMrpDataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortMrpDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortMrpDataAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDPortMrpDataAdjust
0x0215	PDPortMrpDataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortMrpDataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDPortMrpDataReal

Value (hexadecimal)	Used for	Used by
0x0216	MrpManagerParams Sub block for media redundancy manager parameters	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataAdjust.MrpManagerParams PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceMrpDataAdjust.MrpManagerParams PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataReal.MrpManagerParams
0x0217	MrpClientParams Sub block for media redundancy client parameters	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataAdjust.MrpClientParams PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceMrpDataAdjust.MrpClientParams PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataReal.MrpClientParams
0x0219	MrpRingStateData Sub block for media redundancy ring state data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataReal.MrpRingStateData
0x021B	AdjustLinkState Sub block for adjusting LinkState	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataAdjust.AdjustLinkState PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust.AdjustLinkState
0x021C	CheckLinkState Sub block for checking LinkState	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataCheck.CheckLinkState PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust.CheckLinkState
0x021E	CheckSyncDifference Sub block for checking local and remote CableDelay detected by LLDP to discover a sync difference	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataCheck.CheckSyncDifference PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataCheck.CheckSyncDifference
0x021F	CheckMAUTypeDifference Sub block for checking local and remote MAUTypes detected by LLDP	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataCheck.CheckMAUTypeDifference PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataCheck.CheckMAUTypeDifference
0x0220	PDPortFODataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortFODataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDPortFODataReal
0x0221	FiberOpticManufacturerSpecific Sub block for reading real fiber optic manufacturer specific data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortFODataReal.FiberOpticManufacturerSpecific PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDPortFODataReal.FiberOpticManufacturerSpecific

Value (hexadecimal)	Used for	Used by
0x0222	PDPortFODataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortFODataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortFODataAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDPortFODataAdjust
0x0223	PDPortFODataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortFODataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortFODataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDPortFODataCheck
0x0224	AdjustPeerToPeerBoundary Sub block for adjusting the peer to peer boundary	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataAdjust.AdjustPeerToPeerBoundary PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust.AdjustPeerToPeerBoundary
0x0225	AdjustDCPBoundary Sub block for adjusting the DCP boundary	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataAdjust.AdjustDCPBoundary PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust.AdjustDCPBoundary
0x0226	AdjustPreambleLength Sub block for adjusting the used length of preamble	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataAdjust.AdjustPreambleLength PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust.AdjustPreambleLength
0x0227	CheckMAUTypeExtension Sub block for checking MAUTypeExtension	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataCheck. CheckMAUTypeExtension PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataCheck. CheckMAUTypeExtension
0x0228	FiberOpticDiagnosisInfo Sub block for reading real fiber optic diagnosis data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortFODataReal.FiberOpticDiagnosisInfo PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.FiberOpticDiagnosisInfo
0x0229	AdjustMAUTypeExtension Sub block for adjusting MAUTypeExtension	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataAdjust. AdjustMAUTypeExtension PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortDataAdjust. AdjustMAUTypeExtension

Value (hexadecimal)	Used for	Used by
0x022A	PDIRSubframeData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDIRSubframeData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDIRSubframeData PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PdevData.PDIRSubframeData
0x022B	SubframeBlock Sub block for the persistent portion of DFP	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDIRSubframeData.SubframeBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDIRSubframeData.SubframeBlock PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PdevData.PDIRSubframeData.SubframeBlock
0x022C	PDPortDataRealExtended	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataRealExtended PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDPortDataRealExtended
0x022D	PDTimeData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDTimeData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDTimeData
0x022E	PDPortSFPDataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortSFPDataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortSFPDataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDPortSFPDataCheck
0x22F	Reserved	—
0x0230	PDNCDataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDNCDataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDNCDataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDNCDataCheck

Value (hexadecimal)	Used for	Used by
0x0231	MrpInstanceDataAdjustBlock Sub block of PDInterfaceMrpDataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceMrpDataAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDInterfaceMrpDataAdjust
0x0232	MrpInstanceDataRealBlock Sub block of PDInterfaceMrpDataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDInterfaceMrpDataReal
0x0233	MrpInstanceDataCheckBlock Sub block of PDInterfaceMrpDataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceMrpDataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceMrpDataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDInterfaceMrpDataCheck
0x0234	PDPortMrpIcDataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortMrpIcDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortMrpIcDataAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDPortMrpIcDataAdjust
0x0235	PDPortMrpIcDataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortMrpIcDataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDPortMrpIcDataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDPortMrpIcDataCheck
0x0236	PDPortMrpIcDataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortMrpIcDataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDPortMrpIcDataReal
0x0237 – 0x23F	Reserved	—
0x0240	PDInterfaceDataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceDataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDInterfaceDataReal
0x0241	PDRsiInstances	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRsiInstances

Value (hexadecimal)	Used for	Used by
0x0242 – 0x024F	Reserved	—
0x0250	PDInterfaceAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDInterfaceAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDInterfaceAdjust
0x0251	PDPortStatistic	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortStatistic PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDPortStatistic
0x0252 – 0x025F	Reserved	—
0x0260	OwnPort Sub block of PDPortDataRealExtended	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataReal.OwnPort PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDPortDataRealExtended.OwnPort
0x0261	Neighbors Sub block of PDPortDataRealExtended	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDPortDataRealExtended.Neighbors PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.PDPortDataRealExtended.Neighbors
0x0262 – 0x026F	Reserved	—
0x0270	CIMNetConfDataReal	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataReal
0x0271	CIMNetConfDataAdjust	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfDataAdjust
0x0272	CIMStationPortConfigBlock Sub block of CIMNetConfDataReal Sub block of CIMNetConfDataAdjust	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataReal.CIMStationPortConfigBlock PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataAdjust.CIMStationPortConfigBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfDataAdjust.CIMStationPortConfigBlock

Value (hexadecimal)	Used for	Used by
0x0273	CIMStationQueueConfigBlock Sub block of CIMNetConfDataReal Sub block of CIMNetConfDataAdjust	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataReal.CIMStationPortConfigBlock.CIMStationQueueConfigBlock PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataAdjust.CIMStationPortConfigBlock.CIMStationQueueConfigBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfDataAdjust.CIMStationPortConfigBlock.CIMStationQueueConfigBlock
0x0274	NMEDomainTimeDataBlock Sub block of CIMNetConfDataReal Sub block of CIMNetConfDataAdjust	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataReal.NMEDomainTimeDataBlock PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataAdjust.NMEDomainTimeDataBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfDataAdjust.NMEDomainTimeDataBlock
0x0275	CIMNetConfStreamPathData	PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfStreamPathData
0x0276	CIMNetConfSyncTreeData	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfSyncTreeData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfSyncTreeData
0x0277	CIMNetConfUploadNetworkAttributes	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfUploadNetworkAttributes
0x0278	CIMStationForwardingDelayBlock Sub block for CIMNetConfUploadNetworkAttributes Sub block of CIMNetConfExpectedNetworkAttributes	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfUploadNetworkAttributes.CIMStationForwardingDelayBlock PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfExpectedNetworkAttributes.CIMStationForwardingDelayBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfExpectedNetworkAttributes.CIMStationForwardingDelayBlock
0x0279	CIMNetConfExpectedNetworkAttributes	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.CIMNetConfExpectedNetworkAttributes PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfExpectedNetworkAttributes
0x027A	CIMNetConfStreamPathDataReal	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfStreamPathDataReal

Value (hexadecimal)	Used for	Used by
0x027B	CIMStationPortIngressRateLimiterBlock Sub block of CIMNetConfDataReal Sub block of CIMNetConfDataAdjust	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataReal.CIMStationPortConfigBlock.CIMStationPortIngressRateLimiterBlock PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataAdjust.CIMStationPortConfigBlock.CIMStationPortIngressRateLimiterBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfDataAdjust.CIMStationPortConfigBlock.CIMStationPortIngressRateLimiterBlock
0x027C	CIMStationEgressRateLimiterBlock Sub block of CIMNetConfDataReal Sub block of CIMNetConfDataAdjust	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataReal.CIMStationPortConfigBlock.CIMStationEgressRateLimiterBlock PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataAdjust.CIMStationPortConfigBlock.CIMStationEgressRateLimiterBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfDataAdjust.CIMStationPortConfigBlock.CIMStationEgressRateLimiterBlock
0x027D	CIMStationPortCapabilitiesBlock Sub block of CIMNetConfUploadNetworkAttributes Sub block of CIMNetConfExpectedNetworkAttributes	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfUploadNetworkAttributes.CIMStationPortCapabilitiesBlock PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfExpectedNetworkAttributes.CIMStationPortCapabilitiesBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfExpectedNetworkAttributes.CIMStationPortCapabilitiesBlock
0x027E	CIMStationExpectedNeighborBlock Sub block of CIMNetConfExpectedNetworkAttributes	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfExpectedNetworkAttributes.CIMStationExpectedNeighborBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMNetConfExpectedNetworkAttributes.CIMStationExpectedNeighborBlock
0x027F	NMEDomainConfigRealBlock Sub block of CIMNetConfDataReal	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMNetConfDataReal.NMEDomainConfigRealBlock
0x0280 – 0x02FF	Reserved	—
0x0300	CIMSNMPAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.CIMSNMPAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CIMSNMPAdjust
0x0301	CIMElectricPowerReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.CIMElectricPowerReal
0x0302 – 0x03FF	Reserved	—

Value (hexadecimal)	Used for	Used by
0x0400	MultipleBlockHeader	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDRealData.MultipleBlockHeader PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.MultipleBlockHeader
0x0401	COContainerContent	PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.CombinedObjectContainer.COContainerContent
0x0402 – 0x04FF	Reserved	—
0x0500	RecordDataReadQuery	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataReadQuery
0x0501	UNIAddStreamReq	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataReadQuery.UNIAddStreamReq
0x0502	UNIAddStreamRsp	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.UNIAddStreamRsp
0x0503	UNIRemoveStreamReq	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataReadQuery.UNIRemoveStreamReq
0x0504	UNIRemoveStreamRsp	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.UNIRemoveStreamRsp
0x0505	UNIRenewStreamReq	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataReadQuery.UNIRenewStreamReq
0x0506	UNIRenewStreamRsp	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.UNIRenewStreamRsp
0x0507 – 0x050F	Reserved	—
0x0510	CIMDCPGetBlockReq	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataReadQuery.CIMDCPGetBlockReq
0x0511	CIMDCPGetBlockRsp	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMDCPGetBlockRsp
0x0512	CIMDCPSetBlockReq	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataReadQuery.CIMDCPSetBlockReq
0x0513	CIMDCPSetBlockRsp	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.CIMDCPSetBlockRsp
0x0514 – 0x05FF	Reserved	—
0x0600	FSHelloBlock	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceFSUDataAdjust.FSHelloBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceFSUDataAdjust.FSHelloBlock PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDInterfaceFSUDataAdjust.FSHelloBlock
0x0601	FSParameterBlock	PROFINETIO-ServiceReqPDU.IODConnectReq.IOCRBlockReq.ARFSUBlock.ARFSUDataAdjust.FSParameterBlock

Value (hexadecimal)	Used for	Used by
0x0602	FastStartUpBlock	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceFSUDataAdjust.FastStartUpBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceFSUDataAdjust.FastStartUpBlock PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDInterfaceFSUDataAdjust.FastStartUpBlock PROFINETIO-ServiceReqPDU.IODConnectReq.IOCRBlockReq.ARFSUBlock.ARFSUDataAdjust.FastStartUpBlock PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.ARFSUDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.ARFSUDataAdjust PROFINETIO-ServiceReqPDU.IODConnectReq.ARFSUBlock.ARFSUDataAdjust
0x0603 – 0x0607	Reserved for FastStartUp	—
0x0608	PDInterfaceFSUDataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDInterfaceFSUDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDInterfaceFSUDataAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDInterfaceFSUDataAdjust
0x0609	ARFSUDataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.ARFSUDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.ARFSUDataAdjust PROFINETIO-ServiceReqPDU.IODConnectReq.ARFSUBlock.ARFSUDataAdjust
0x060A – 0x060F	Reserved for FastStartUp	—
0x0610 – 0x06FF	Reserved	—
0x0700 – 0x07FF	Reserved	—
0x0800	Reserved for profiles covering energy saving BlockType for request service	PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite
0x0801	Reserved for profiles covering energy saving BlockType for response service	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead
0x0802 – 0x080F	Reserved for profiles covering energy saving	—
0x0810	PE_EntityFilterData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PE_EntityFilterData

Value (hexadecimal)	Used for	Used by
0x0811	PE_EntityStatusData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PE_EntityStatusData
0x0812 – 0x087F	Reserved for profiles covering energy saving	—
0x0880 – 0x08FF	Reserved	—
0x0900	RS_AdjustObserver	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.RS_AdjustObserver PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.RS_AdjustObserver
0x0901	RS_GetEvent	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.RS_GetEvent
0x0902	RS_AckEvent	PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.RS_AckEvent
0x0903 – 0x097F	Reserved for reporting system	—
0x0980 – 0x09FF	Reserved	—
0x0A00	Upload BLOB Query	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.UploadBLOBQuery
0x0A01	Upload BLOB	PROFINETIO-ServiceResPDU.IODReadRes.RecordDataRead.UploadBLOB
0x0A02 – 0x0AFF	Reserved for profiles covering controller to controller communication	—
0x0B00 – 0x0EFF	Reserved	—
0x0F00	MaintenanceItem	RTA-SDU.AlarmNotification-PDU.AlarmPayload.MaintenanceItem
0x0F01	Upload selected records within Upload&RetrievalItem	RTA-SDU.AlarmNotification-PDU.AlarmPayload.Upload&RetrievalItem
0x0F02	iParameterItem	RTA-SDU.AlarmNotification-PDU.AlarmPayload.iParameterItem
0x0F03	Retrieve selected records within Upload&RetrievalItem	RTA-SDU.AlarmNotification-PDU.AlarmPayload.Upload&RetrievalItem
0x0F04	Retrieve all stored records within Upload&RetrievalItem	RTA-SDU.AlarmNotification-PDU.AlarmPayload.Upload&RetrievalItem
0x0F05	Signal a PE_OperationalMode change within PE_EnergySavingStatus	RTA-SDU.AlarmNotification-PDU.AlarmPayload.PE_AlarmItem
0x0F06 – 0x6FEF	Reserved	—
0x6FF0 – 0x6FFF	Reserved for conformance testing purpose	Administrative numbers assigned to conformance testing purpose
0x7000 – 0x7FFF	Reserved for vendor specific data objects	Administrative numbers assigned to vendor specific defined data objects
0x8000	Reserved	—
0x8001	Alarm Ack High	RTA-SDU.AlarmAck-PDU
0x8002	Alarm Ack Low	RTA-SDU.AlarmAck-PDU
0x8008	IODWriteResHeader	PROFINETIO-ServiceResPDU.IODWriteRes
0x8009	IODReadResHeader	PROFINETIO-ServiceReqPDU.IODReadRes
0x800A – 0x8100	Reserved	—

Value (hexadecimal)	Used for	Used by
0x8101	ARBlockRes	PROFINETIO-ServiceResPDU.IODConnectRes.ARBlockRes
0x8102	IOCRBlockRes	PROFINETIO-ServiceResPDU.IODConnectRes.IOCRBlockRes
0x8103	AlarmCRBlockRes	PROFINETIO-ServiceResPDU.IODConnectRes.AlarmCRBlockRes
0x8104	ModuleDiffBlock	PROFINETIO-ServiceResPDU.IODConnectRes.ModuleDiffBlock PROFINETIO-ServiceResPDU.IODWriteRes.RecordDataRead.ModuleDiffBlock PROFINETIO-ServiceReqPDU.IOXControlReq.ModuleDiffBlock
0x8105	PrmServerBlockRes	PROFINETIO-ServiceResPDU.IODConnectRes.PrmServerBlockRes
0x8106	ARServerBlockRes	PROFINETIO-ServiceResPDU.IODConnectRes.ARServerBlockRes
0x8107	ARRPCBlockRes	PROFINETIO-ServiceResPDU.IODConnectRes.ARRPCBlockRes
0x8108	ARVendorBlockRes	PROFINETIO-ServiceResPDU.IODConnectRes.ARVendorBlockRes
0x8109 – 0x810F	Reserved	—
0x8110	IODBlockRes, shall only be used in conjunction with connection establishment phase	PROFINETIO-ServiceResPDU.IODControlRes.ControlBlockConnect (PrmEnd).rsp
0x8111	IODBlockRes, shall only be used in conjunction with a plug alarm event	PROFINETIO-ServiceResPDU.IODControlRes.ControlBlockPlug (PrmEnd).rsp
0x8112	IOXBlockRes, shall only be used in conjunction with connection establishment phase	PROFINETIO-ServiceResPDU.IOXControlRes.ControlBlockConnect (ApplRdy).rsp
0x8113	IOXBlockRes, shall only be used in conjunction with a plug alarm event	PROFINETIO-ServiceResPDU.IOXControlRes.ControlBlockPlug (ApplRdy).rsp
0x8114	ReleaseBlockRes	PROFINETIO-ServiceResPDU.IODReleaseRes.ReleaseBlock
0x8115	Reserved	—
0x8116	IOXBlockRes, shall only be used in conjunction with connection establishment phase	PROFINETIO-ServiceResPDU.IOXControlRes.ControlBlockConnect (ReadyForCompanion).rsp
0x8117	IOXBlockRes, shall only be used in conjunction with connection establishment phase	PROFINETIO-ServiceResPDU.IOXControlRes.ControlBlockConnect (ReadyForRT_CLASS_3).rsp
0x8118	IODBlockRes	PROFINETIO-ServiceResPDU.IODControlRes.ControlBlockConnect (PrmBegin).rsp
0x8119	Reserved	—
0x811A	SecurityResponseBlock	Reserved for security
Other	Reserved	—

5.2.1.2 Coding of the field BlockLength

This field shall be coded as data type Unsigned16 with the values according to Table 544.

Table 544 – BlockLength

Value (hexadecimal)	Meaning
0x0000 – 0x0002	Reserved
0x0003 – 0xFFFF	Number of octets without counting the fields BlockType and BlockLength

5.2.1.3 Coding of the field BlockVersionHigh

This field shall be coded as data type Unsigned8 with the values according to Table 545.

Table 545 – BlockVersionHigh

Value (hexadecimal)	Meaning	Use
0x00	Reserved	—
0x01	Version 1	Indicates version 1
0x02 – 0xFF	Reserved	—

5.2.1.4 Coding of the field BlockVersionLow

This field shall be coded as data type Unsigned8 with the values according to Table 546.

Table 546 – BlockVersionLow

Value (hexadecimal)	Meaning	Use
0x00	Version 0	Indicates version 0
0x01	Version 1	Indicates version 1
0x02 – 0xFF	Version 2 to version 255	Indicates version 2 to 255

5.2.2 Coding section related to RTA-SDU specific fields

5.2.2.1 Coding of the field AlarmType

This field shall be coded as data type Unsigned16 with the values according to Table 547. AlarmType and UserStructureIdentifier combinations are shown in Table 711.

Table 547 – AlarmType

Value (hexadecimal)	Meaning	Alarm priority	Attached to the Diagnosis ASE	AlarmPayload required	USI structured AlarmPayload	Allowed USI range
0x0000	Reserved	—	—	—	—	—
0x0001	Diagnosis	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x0002	Process	Alarm_High	No	Optional	Yes	Manufacturer specific Normative Profile specific
0x0003	Pull [a]	Alarm_Low	No	Optional	Yes	Normative
0x0004	Plug	Alarm_Low	No	Optional	Yes	Normative
0x0005	Status	Alarm_Low	No	Optional	Yes	Manufacturer specific Normative Profile specific
0x0006	Update	Alarm_Low	No	Optional	Yes	Manufacturer specific Normative Profile specific
0x0007	Media Redundancy Both Ring and Interconnection	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x0008	Controlled by supervisor Logical “Pull” of a submodule to withdraw ownership	Alarm_Low	No	Optional	Yes	Normative
0x0009	Released Logical “Plug” of a submodule to return ownership or trigger a reparameterization	Alarm_Low	No	Optional	Yes	Normative

Value (hexadecimal)	Meaning	Alarm priority	Attached to the Diagnosis ASE	AlarmPayload required	USI structured AlarmPayload	Allowed USI range
0x000A	Plug Wrong Submodule [c]	Alarm_Low	No	Optional	Yes	Normative
0x000B	Return of Submodule [e]	Alarm_Low	No	Optional	Yes	Normative
0x000C	Diagnosis disappears	Alarm_Low	Yes	Optional [d]	Yes	Manufacturer specific Normative Profile specific
0x000D	Multicast communication mismatch notification	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x000E	Port data change notification	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x000F	Sync data change notification	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x0010	Isochronous mode problem notification	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x0011	Network component problem notification	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x0012	Time data change notification	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x0013	Dynamic Frame Packing problem notification	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x0014	MRPD problem notification	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x0015	Reserved	—	—	—	—	—

Value (hexadecimal)	Meaning	Alarm priority	Attached to the Diagnosis ASE	AlarmPayload required	USI structured AlarmPayload	Allowed USI range
0x0016	Multiple interface mismatch notification	Alarm_Low	Yes	Yes	Yes	Manufacturer specific Normative Profile specific
0x0017 – 0x001D	Reserved	—	—	—	—	—
0x001E	Upload and retrieval notification	Alarm_Low	No	Yes	Yes	Manufacturer specific Normative Profile specific
0x001F	Pull module [b]	Alarm_Low	No	Optional	Yes	Normative
0x0020 – 0x007F	Manufacturer specific	Alarm_Low	No	Optional	Yes	Manufacturer specific
0x0080 – 0x00FF	Reserved for profiles	Alarm_Low	No	Optional	Yes	Profile specific
0x0100 – 0xFFFF	Reserved	—	—	—	—	—

[a] With ARProperties.PullModuleAlarmAllowed(=0) SubslotNumber 0x0001 – 0x9FFF used as “Pull submodule” and SubslotNumber := 0 used as “Pull module”.

[b] With ARProperties.PullModuleAlarmAllowed(=1) AlarmType(Pull) shall signal pulling of submodule and AlarmType(Pull module) shall signal pulling of module.

[c] Only an information for the application of the CMCTL.

[d] Without AlarmPayload the signaling Submodule indicates that all earlier reported entries to the Diagnosis ASE are gone.

[e] Only this AlarmType is used for SharedInput submodules, too.

5.2.2.2 Coding of the field AlarmSpecifier

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 10: AlarmSpecifier.SequenceNumber
This field shall be coded according to Table 548. By means of the SequenceNumber the receiver detects duplications and reflects the value of the field in the AlarmAck.

Each alarm priority (HIGH or LOW) administrates its own SequenceNumber. It should start with zero and shall be incremented by one with every AlarmNotification. The Alarm receiver shall accept an arbitrary value as start value.

NOTE The SequenceNumber can start for each AR of an ARset with an arbitrary value at every switchover.

Whenever an Alarm is repeated via different ARs of an ARset the same SequenceNumber shall be used.

Table 548 – AlarmSpecifier.SequenceNumber

Value (hexadecimal)	Meaning
0x0000 – 0x07FF	Allowed value range

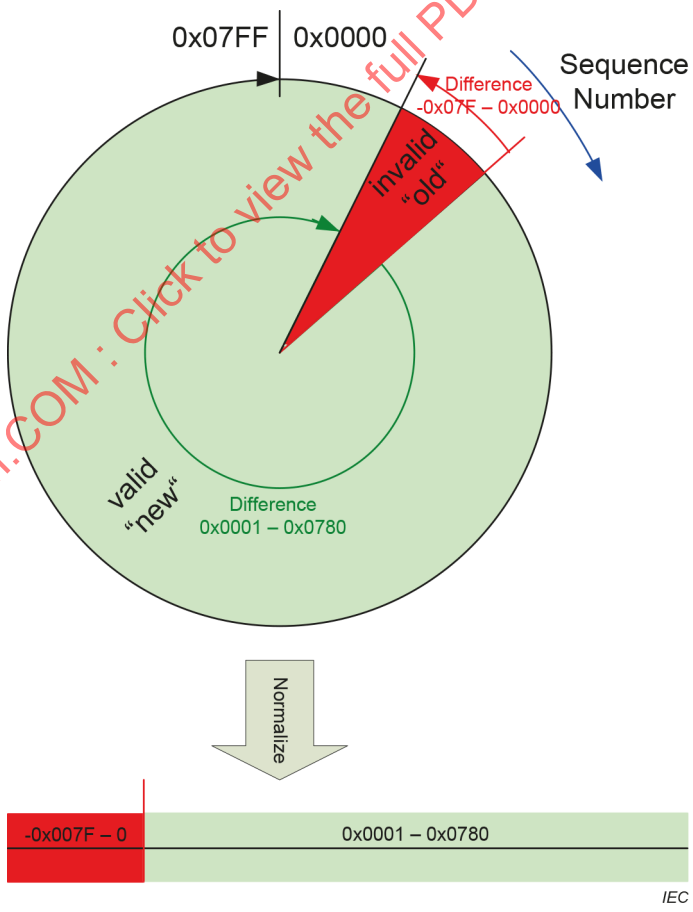


Figure 171 – AlarmSpecifier.SequenceNumber value range

Table 549 defines a maximum SequenceNumber window between an old and a new alarm.

Table 549 – AlarmSpecifier.SequenceNumber Difference

Value (hexadecimal)	Meaning	Use
-0x007F – 0x0000	Red sector according to Figure 171 1/16 of the SequenceNumber range The alarm sink votes the received alarm as “older”	Alarm is dropped
0x0001 – 0x0780	Green sector according to Figure 171 15/16 of the SequenceNumber range The alarm sink votes the received alarm as “newer”	Alarm is processed

Bit 11: AlarmSpecifier.ChannelDiagnosis
This field shall be coded with the values according to Table 550. For all other AlarmTypes this field shall be set to zero.

Table 550 – AlarmSpecifier.ChannelDiagnosis

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Means that the Submodule (alarm source) contains no ChannelDiagnosis, ExtChannelDiagnosis or QualifiedChannelDiagnosis with Channel-Properties.Maintenance (=Fault) and Channel-Properties.Specifier (=appear)	All Diagnosis ASE attached AlarmTypes
0x01	Means that the Submodule (alarm source) contains at least one ChannelDiagnosis, ExtChannelDiagnosis or QualifiedChannelDiagnosis with Channel-Properties.Maintenance (=Fault) and Channel-Properties.Specifier (=appear)	

Bit 12: AlarmSpecifier.ManufacturerSpecificDiagnosis
This field shall be coded with the values according to Table 551. For all other AlarmTypes this field shall be set to zero.

Table 551 – AlarmSpecifier.ManufacturerSpecificDiagnosis

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Means that the Submodule (alarm source) contains no ManufacturerSpecificDiagnosis with Channel-Properties.Maintenance (=Fault) and Channel-Properties.Specifier (=appear or disappear)	All Diagnosis ASE attached AlarmTypes
0x01	Means that the Submodule (alarm source) contains at least one ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance (=Fault) and ChannelProperties.Specifier (=appear or disappear)	

Bit 13: AlarmSpecifier.SubmoduleDiagnosisState
This field shall be coded with the values according to Table 552. For all other AlarmTypes this field shall be set to zero.

Table 552 – AlarmSpecifier.SubmoduleDiagnosisState

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Fault free No DiagnosisData with ChannelProperties.Maintenance (=Fault) and ChannelProperties.Specifier (=appear) exists at the submodule. Furthermore, it indicates that all reported diagnosis has been cleared. An individual “disappears” notification can be omitted. However, even in this case a Manufacturer Specific Diagnosis with ChannelProperties.Maintenance (=Fault) and ChannelProperties.Specifier (=disappear) may be present.	All Diagnosis ASE attached AlarmTypes
0x01	Fault exists At least one DiagnosisData with Channel-Properties.Maintenance (=Fault) and Channel-Properties.Specifier (=appear) exists at the submodule. This bit summarizes the Fault information in all codings. See Table 754	

Bit 14: AlarmSpecifier.Reserved
This field shall be set to zero.

Bit 15: AlarmSpecifier.ARDiagnosisState
This field shall be coded with the values according to Table 553. For all other AlarmTypes this field shall be set to zero.

Table 553 – AlarmSpecifier.ARDiagnosisState

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Fault free No DiagnosisData with ChannelProperties.Maintenance(=Fault) and ChannelProperties.Specifier(=appear) exists at the AR. Furthermore, it indicates that all reported diagnosis has been cleared. An individual “disappears” notification can be omitted. However, even in this case a Manufacturer Specific Diagnosis with ChannelProperties.Maintenance(=Fault) and ChannelProperties.Specifier(=disappear) may be present.	All Diagnosis ASE attached AlarmTypes
0x01	Fault exists At least one DiagnosisData with ChannelProperties.Maintenance(=Fault) and ChannelProperties.Specifier(=appear) exists at the AR.	

5.2.3 Coding section related to common address fields

5.2.3.1 Coding of the field API

This field, defined as an administrative number, shall be coded as data type Unsigned32 according to Table 554.

Table 554 – API

Value (hexadecimal)	Meaning of API
0	Default
0x00000001 – 0x0000FFFF	Used for profile definitions fitting for 5.2.7.9
0x00010000 – 0x0001FFFF	Used for profile definitions beyond 5.2.7.9 Assigned to manufacturer identified by VendorID as shown in Formula (52).
0x00020000 – 0xFFFFFFFF	Used for profile definitions beyond 5.2.7.9

NOTE API means Application Process Identifier.

$$API = 0x10000 + VendorID \quad (52)$$

where

API is the corresponding value

0x10000 is the basic value for this formula

VendorID is the manufacturer assigned administrative number

5.2.3.2 Coding of the field SlotNumber

This field shall be coded as data type Unsigned16 according to Table 555.

Table 555 – SlotNumber

Value (hexadecimal)	Meaning of SlotNumber
0 – 0x7FFF	The first usable slot for modules is zero. The last usable slot for modules is 0x7FFF. It may contain gaps.
0x8000	Used for the Alarm transport container. Used in conjunction with reporting system.
0x8001 – 0xFFFF	Reserved

5.2.3.3 Coding of the field SubslotNumber

This field shall be coded as data type Unsigned16 according to Table 556.

Table 556 – SubslotNumber

Value (hexadecimal)	Meaning of SubslotNumber	
0x0000	ARProperties.PullModuleAlarmAllowed (=0)	Shall be used to code “Pull module” in conjunction with AlarmType (Pull).
	ARProperties.PullModuleAlarmAllowed (=1)	Usable subslot. The first usable subslot for submodules is zero.
0x0001 – 0x7FFF	The last usable subslot for submodules is 0x7FFF. There may be gaps.	
0x8000 – 0x8FFF	Default Used for 16 interface submodules (each bound to one end station component) with up to 255 (end station component or bridge component) ports. There may be gaps. See Formula (53) If used by system redundancy, the interface submodule mounted from left or top in the rack.	
0x9000 – 0x9FFF	Used for 16 interface submodules (each bound to one end station component) with up to 255 (end station component or bridge component) ports. There may be gaps. See Formula (53) Used by system redundancy with redundant interface for the interface submodule mounted from right or bottom in the rack.	
0xA000 – 0xA00F	Partnering with 0x8000 – 0x8FFF Bridge component Addressing up to 16 bridge components in a device	
0xA010 – 0xA01F	Reserved for bridge component	
0xA020 – 0xA02F	Partnering with 0x8000 – 0x8FFF Bridge component port Addressing up to 16 bridge component ports of the above addressed bridge components	
0xA030 – 0xA03F	Reserved for bridge component port	
0xA040 – 0xA04F	Partnering with 0x8000 – 0x8FFF End station component port Addressing up to 16 end station component ports internally connected to a bridge component Example: 0xA040 bound to 0x8000 with interface 1 connected to end station component 1 port 1 … 0xA04F bound to 0x8F00 with interface 16 connected to end station component 16 port 1	
0xA050 – 0xA05F	Reserved for end station component port	
0xA060 – 0xA07F	Reserved for components	
0xA080 – 0xA08F	Partnering with 0x9000 – 0x9FFF Bridge component Addressing up to 16 bridge components in a device	
0xA090 – 0xA09F	Reserved for bridge component	
0xA0A0 – 0xA0AF	Partnering with 0x9000 – 0x9FFF Bridge component port Addressing up to 16 bridge component ports of the above addressed bridge components	
0xA0B0 – 0xA0BF	Reserved for bridge component port	

Value (hexadecimal)	Meaning of SubslotNumber
0xA0C0 – 0xA0CF	Partnering with 0x9000 – 0x9FFF End station component port Addressing up to 16 end station component ports internally connected to a bridge component Example: 0xA0C0 bound to 0x9000 with interface 1 connected to end station component 1 port 1 … 0xA0CF bound to 0x9F00 with interface 16 connected to end station component 16 port 1
0xA0D0 – 0xA0DF	Reserved for end station component port
0xA0E0 – 0xA0FF	Reserved for components
0xA100 – 0xFFFF	Reserved

The subslotnumber shall be evaluated with 0xLIPP with

– L counting NAPs,
– I counting interfaces (PP == 00 means interface itself), and
– PP counting ports

as shown in Formula (53).

$$\text{SubslotNumber} = 0xL000 + 0x0I00 + 0x00PP \quad (53)$$

where

SubslotNumber	is the corresponding value
L	is the basic value for this formula 0x8000 or 0x9000
I	is the number of the interface starting with zero
PP	is the number of the external visible port of an interface starting with the value one. The value zero identifies the end station component bound to the interface.

The IEEE Std 802.1Q model for end stations and bridges is translated into this addressing scheme. Thus, internal ports used to connect end stations and bridge components needs to be identified and addressable as well as external ports and interfaces.

5.2.3.4 Coding of the field Index

5.2.3.4.1 General

This field shall be coded as data type Unsigned16 according to Table 557. It is defined for each section whether the fields TargetARUUID, ARUUID, SlotNumber and SubslotNumber shall be ignored or used.

Furthermore, all services with write access shall only use the valid address space of the established AR context. Read services may address record data beyond this context.

Table 557 – Index range

Index (hexadecimal)	Comment
0x0000 – 0x7FFF	User specific Record address space
0x8000 – 0xBFFF	Normative subslot specific with profile section Protocol specific function
0xC000 – 0xDFFF	Normative slot specific with profile section Protocol specific function
0xE000 – 0xEFFF	Normative AR specific with profile section Protocol specific function
0xF000 – 0xF7FF	Normative API specific with profile section Protocol specific function
0xF800 – 0xFFFF	Normative device specific with profile section Protocol specific function

5.2.3.4.2 Usage of the address parameters

The following expressions show the evaluation of the address parameters according to the index range.

The rules for evaluating the address parameter as shown in Table 558, Table 559, Table 560, Table 561, and Table 562 shall be applied.

NOTE Implicit means from outside an existing AR. Explicit means from inside the existing AR, including DeviceAccess.

Table 558 – Expression 1 (subslot specific)

RPCOperationNmb	ARUUID	Target-ARUUID	Validity
Implicit Read	NIL (implicit AR)	Evaluated if needed by the index	API, Slot Number, Subslot Number, Index shall be evaluated
Explicit Read/Write	Not NIL (explicit AR)	Ignore	API, Slot Number, Subslot Number, Index shall be evaluated

Table 559 – Expression 2 (slot specific)

RPCOperationNmb	ARUUID	Target-ARUUID	Validity
Implicit Read	NIL (implicit AR)	Evaluated if needed by the index	API, Slot Number, Index shall be evaluated
Explicit Read/Write	Not NIL (explicit AR)	Ignore	API, Slot Number, Index shall be evaluated

Table 560 – Expression 3 (AR specific)

RPCOperationNmb	ARUUID	Target-ARUUID	Validity
Implicit Read	NIL (implicit AR)	Evaluated if needed by the index	Index shall be evaluated
Explicit Read/Write	Not NIL (explicit AR)	Ignore	Index shall be evaluated

Table 561 – Expression 4 (API specific)

RPCOperationNmb	ARUUID	Target ARUUID	Validity
Implicit Read	NIL (implicit AR)	Ignore	API, Index shall be evaluated
Explicit Read/Write	Not NIL (explicit AR)	Ignore	API, Index shall be evaluated

Table 562 – Expression 5 (device specific)

RPCOperationNmb	ARUUID	Target-ARUUID	Validity
Implicit Read	NIL (implicit AR)	Ignore	Index shall be evaluated
Explicit Read/Write	Not NIL (explicit AR)	Ignore	Index shall be evaluated

5.2.3.4.3 Grouping of DiagnosisData for the diagnosis records

The diagnosis records contain different data dependent on the index. The grouping shall be according to Table 563. The identifier is used in Table 567, Table 568, Table 569, Table 570 and Table 571.

Table 563 – Grouping of DiagnosisData

Identifier	Meaning
Diagnosis in channel coding	ChannelDiagnosis, ExtChannelDiagnosis and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=Fault) and ChannelProperties.Specifier(=appear)
Diagnosis in all codings	ChannelDiagnosis, ExtChannelDiagnosis, QualifiedChannelDiagnosis and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=Fault) and ChannelProperties.Specifier(=appear)
Maintenance required in channel coding	ChannelDiagnosis, ExtChannelDiagnosis and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=MaintenanceRequired) and ChannelProperties.Specifier(=appear)
Maintenance required in all codings	ChannelDiagnosis, ExtChannelDiagnosis and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=MaintenanceRequired) and ChannelProperties.Specifier(=appear)
Maintenance demanded in channel coding	ChannelDiagnosis, ExtChannelDiagnosis and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=MaintenanceDemanded) and ChannelProperties.Specifier(=appear)
Maintenance demanded in all codings	ChannelDiagnosis, ExtChannelDiagnosis and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=MaintenanceDemanded) and ChannelProperties.Specifier(=appear)
Diagnosis, Maintenance, Qualified and Status	ChannelDiagnosis, ExtChannelDiagnosis, QualifiedChannelDiagnosis, ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=any) and ChannelProperties.Specifier(=appear), and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=Fault) and ChannelProperties.Specifier(=disappear)

5.2.3.4.4 Access control roles

Table 564 and Table 565 show the roles which are assigned to different entities. These roles are checked by the ACD and provided by certificates.

Table 564 – SecurityControlRole

Identifier	Meaning
CredentialManager	Reserved for security
SecurityConfigurationManager	Reserved for security

Table 565 – AccessControlRole

Identifier	Meaning
Controller	Controller AR establishment.
NetworkManagement	Network management Configures the network using the CIM interface.
Diagnostics	Diagnostics Tools reading the status and diagnostic data.
OperatorStation	Operator station Control entity reading and writing data.

5.2.3.4.5 Record index

A NULL response for a read record request shall only be created if

– the defined filter leads to no matching items (for example DiagnosisData)
– the addressed record was not written (for example Adjust or Check)
– the addressed object contains no information (for example LogBookData without entry or RecordInputDataObjectElement for inputs without data)

The coding for user specific record data shall be according to Table 566 and the usage of the address parameters according to Table 558.

Table 566 – Index (user specific)

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0 – 0x7FFF	User specific RecordData	All	—	User specific definition	User specific definition	Diagnostics, User specific definition	Controller, Operator-Station, User specific definition	User specific definition	User specific definition	User specific definition

The coding for submodule specific defined record data shall be according to Table 567 and the usage of the address parameters according to Table 558.

Table 567 – Index (subslot specific)

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0x8000	ExpectedIdentificationData for one subslot	All	Needed	Possible	Read	Diagnostics	—	—	—	No
0x8001	RealIdentificationData for one subslot	All	—	—	Read	Diagnostics	—	—	—	No
0x8002 – 0x8009	Reserved	—	—	—	—	—	—	—	—	—
0x800A	DiagnosisData Diagnosis in channel coding for one subslot	All	Filter	Possible	Read	Diagnostics	—	—	—	No
0x800B	DiagnosisData Diagnosis in all codings for one subslot	All	Filter	Possible	Read	Diagnostics	—	—	—	No
0x800C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one subslot	All	Filter	Possible	Read	Diagnostics	—	—	—	No
0x800D – 0x800F	Reserved	—	—	—	—	—	—	—	—	—
0x8010	DiagnosisData Maintenance required in channel coding for one subslot	All	Filter	Possible	Read	Diagnostics	—	—	—	No
0x8011	DiagnosisData Maintenance demanded in channel coding for one subslot	All	Filter	Possible	Read	Diagnostics	—	—	—	No
0x8012	DiagnosisData Maintenance required in all codings for one subslot	All	Filter	Possible	Read	Diagnostics	—	—	—	No

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0x8013	DiagnosisData Maintenance demanded in all codings for one subslot	All	Filter	Possible	Read	Diagnostics	—	—	—	No
0x8014 – 0x801D	Reserved	—	—	—	—	—	—	—	—	—
0x801E	SubstituteValue for one subslot	All	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	—	Yes	Yes
0x801F	Reserved	—	—	—	—	—	—	—	—	—
0x8020	PDIRSubframeData for one subslot	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	—
0x8021 – 0x8026	Reserved	—	—	—	—	—	—	—	—	—
0x8027	PDPortDataRealExtended for one subslot	Port	—	—	Read	Diagnostics	—	—	—	No
0x8028	RecordInputDataObjectElement for one subslot	All	—	Possible	Read	Diagnostics	—	—	—	Yes
0x8029	RecordOutputDataObjectElement for one subslot	All	—	—	Read	Diagnostics	—	—	—	Yes
0x802A	PDPortDataReal for one subslot	Port	—	—	Read	Diagnostics	—	—	—	No
0x802B	PDPortDataCheck for one subslot	Port	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x802C	PDIRData for one subslot	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	—
0x802D	PDSyncData for one subslot with SyncID value 0	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	—
0x802E	Reserved (legacy)	—	—	—	—	—	—	—	—	—
0x802F	PDPortDataAdjust for one subslot	Port	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8030	IsochronousModeData for one subslot	All	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	—

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0x8031	PDTimeData for one subslot	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8032 – 0x804F	Reserved	—	—	—	—	—	—	—	—	—
0x8050	PDInterfaceMrpDataReal for one subslot	Interface	—	Possible	Read	Diagnostics	—	—	—	No
0x8051	PDInterfaceMrpDataCheck for one subslot	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8052	PDInterfaceMrpDataAdjust for one subslot	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8053	PDPortMrpDataAdjust for one subslot	Port	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8054	PDPortMrpDataReal for one subslot	Port	—	Possible	Read	Diagnostics	—	—	—	No
0x8055	PDPortMrpIcDataAdjust for one subslot	Port	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8056	PDPortMrpIcDataCheck for one subslot	Port	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8057	PDPortMrpIcDataReal for one subslot	Port	—	Possible	Read	Diagnostics	—	—	—	No
0x8058 – 0x805F	Reserved	—	—	—	—	—	—	—	—	—
0x8060	PDPortFODataReal for one subslot	Port	—	—	Read	Diagnostics	—	—	—	No
0x8061	PDPortFODataCheck for one subslot	Port	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8062	PDPortFODataAdjust for one subslot	Port	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8063	PDPortSFPDataCheck for one subslot	Port	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8064 – 0x806F	Reserved	—	—	—	—	—	—	—	—	—
0x8070	PDNCDataCheck for one subslot	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0x8071	PDInterfaceAdjust for one subslot	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x8072	PDPortStatistic for one subslot	Interface, Port	—	—	Read	Diagnostics	—	—	—	No
0x8073 – 0x807F	Reserved	—	—	—	—	—	—	—	—	—
0x8080	PDInterfaceDataReal for one subslot	Interface	—	—	Read	Diagnostics	—	—	—	No
0x8081 – 0x808F	Reserved	—	—	—	—	—	—	—	—	—
0x8090	PDInterfaceFSUDataAdjust	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	—
0x8091 – 0x809F	Reserved	—	—	—	—	—	—	—	—	—
0x80A0	Profiles covering energy saving – Record_0	All	Address	—	Read / Write	Diagnostics	Controller, Operator-Station	—	Yes	Profile
0x80A1 – 0x80AE	Reserved for profiles covering energy saving	—	—	—	—	—	—	—	—	—
0x80AF	PE_EntityStatusData for one subslot	All	—	Possible	Read	Diagnostics	—	—	—	Profile
0x80B0	CombinedObjectContainer [a]	All	—	—	Write	—	—	—	—	—
0x80B1 – 0x80BF	Reserved	—	—	—	—	—	—	—	—	—
0x80C0 – 0x80CE	Reserved for reporting system	—	—	—	—	—	—	—	—	—
0x80CF	RS_AdjustObserver	All	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	Yes
0x80D0	Profiles covering condition monitoring – Record_0	All	—	—	—	—	—	—	—	—
0x80D1 – 0x80DF	Reserved for profiles covering condition monitoring	—	—	—	—	—	—	—	—	—
0x80E0 – 0x80EF	Reserved for profiles covering functional safety	—	—	—	—	—	—	—	—	—

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0x80F0	CIMNetConfDataReal	Interface	—	—	Read	Diagnostics	—	—	—	No
0x80F1	CIMNetConfStreamPathData	Interface	—	—	Write	—	NetworkManagement	Yes	—	No
0x80F2	CIMNetConfSyncTreeData	Interface	—	—	Read / Write	Diagnostics	NetworkManagement	Yes	—	No
0x80F3	CIMNetConfUploadNetworkAttributes	Interface	—	—	Read	Diagnostics	—	—	—	No
0x80F4	CIMNetConfExpectedNetworkAttributes	Interface	—	—	Read / Write	Diagnostics	NetworkManagement	Yes	—	No
0x80F3	CIMNetConfUploadNetworkAttributes	Interface	—	—	Read	Diagnostics	—	—	—	No
0x80F4	CIMNetConfExpectedNetworkAttributes	Interface	—	—	Read / Write	Diagnostics	NetworkManagement	Yes	—	No
0x80F5	CIMNetConfDataAdjust	Interface	—	—	Read / Write	Diagnostics	NetworkManagement	Yes	—	No
0x80F6	CIMNetConfStreamPathDataReal for “Time-aware stream, class HIGH”	Interface	—	—	Read	Diagnostics	—	—	—	No
0x80F7	CIMNetConfStreamPathDataReal for “Time-aware stream, class HIGH redundant”	Interface	—	—	Read	Diagnostics	—	—	—	No
0x80F8	CIMNetConfStreamPathDataReal for “Time-aware stream, class LOW”	Interface	—	—	Read	Diagnostics	—	—	—	No
0x80F9	CIMNetConfStreamPathDataReal for “Time-aware stream, class LOW redundant”	Interface	—	—	Read	Diagnostics	—	—	—	No
0x80FA – 0x80FF	Reserved for CIM data	—	—	—	—	—	—	—	—	—
0x8100 – 0x81FF	Reserved	—	—	—	—	—	—	—	—	—
0x8200	CIMSNMPAdjust	Interface	—	Possible	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	No
0x8201 – 0xAFEF	Reserved	—	—	—	—	—	—	—	—	—
0xAFF0	I&M0	All	—	—	Read	Diagnostics	—	—	—	No

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xAFF1	I&M1	All	—	Possible [b]	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	No
0xAFF2	I&M2	All	—	Possible [b]	Read / Write	Diagnostics	Controller, Operator-Station	Yes	—	No
0xAFF3	I&M3	All	—	Possible [b]	Read / Write	Diagnostics	Controller, Operator-Station	Yes	Yes	No
0xAFF4	I&M4	All	—	Possible [b]	Read / Write	Diagnostics	Controller, Operator-Station	Yes	—	No
0xAFF5	I&M5	All	—	—	Read	Diagnostics	—	—	—	No
0xAFF6 – 0xAFFF	I&M6 – I&M15 [c]	All	—	—	—	Diagnostics	Controller, Operator-Station	—	—	No
0xB000 – 0xBFFF	Reserved for profiles	All	—	—	—	Diagnostics	Controller, Operator-Station	—	—	—

Needed:
Reference to an AR needed.
Any service without this reference shall return an error.

Filter:
AR context, ExpectedIdentification, is used as filter for the response.
Any service without this reference returns data according RealIdentification.

Address:
AR context is used as additional address information for request and response.

[a] The ACD Roles for the contained records apply.

[b] See 5.2.7.2.

[c] Reserved for additional identification and maintenance data.

The coding for module specific defined record data shall be according to Table 568 and the usage of the address parameters according to Table 559.

Table 568 – Index (slot specific)

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xC000	ExpectedIdentificationData for one slot	Needed	Possible	Read	Diagnostics	—	—	—	No
0xC001	RealIdentificationData for one slot	Filter [a]	—	Read	Diagnostics	—	—	—	No
0xC002 – 0xC009	Reserved	—	—	—	—	—	—	—	—
0xC00A	DiagnosisData Diagnosis in channel coding for one slot	Filter	Possible	Read	Diagnostics	—	—	—	No
0xC00B	DiagnosisData Diagnosis in all codings for one slot	Filter	Possible	Read	Diagnostics	—	—	—	No
0xC00C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one slot	Filter	Possible	Read	Diagnostics	—	—	—	No
0xC00D – 0xC00F	Reserved	—	—	—	—	—	—	—	—
0xC010	DiagnosisData Maintenance required in channel coding for one slot	Filter	Possible	Read	Diagnostics	—	—	—	No
0xC011	DiagnosisData Maintenance demanded in channel coding for one slot	Filter	Possible	Read	Diagnostics	—	—	—	No
0xC012	DiagnosisData Maintenance required in all codings for one slot	Filter	Possible	Read	Diagnostics	—	—	—	No
0xC013	DiagnosisData Maintenance demanded in all codings for one slot	Filter	Possible	Read	Diagnostics	—	—	—	No
0xC014 – 0xCFFF	Reserved	—	—	—	—	—	—	—	—

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xD000 – 0xDFFF	Reserved for profiles	—	—	—	Diagnostics	—	—	—	—
Needed: Reference to an AR needed. Any service without this reference shall return an error. Filter: AR context, ExpectedIdentification, is used as filter for the response. Any service without this reference returns data according RealIdentification.									
[a] Even in AR context, ExpectedIdentification may be ignored for the response.									

The coding for AR specific defined record data shall be according to Table 569 and the usage of the address parameters according to Table 560.

Table 569 – Index (AR specific)

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xE000	ExpectedIdentificationData for one AR	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE001	RealIdentificationData for one AR	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE002	ModuleDiffBlock for one AR	Needed	Possible	Read	Diagnostics	—	—	—	Yes
0xE003 – 0xE009	Reserved	—	—	—	—	—	—	—	—
0xE00A	DiagnosisData Diagnosis in channel coding for one AR	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE00B	DiagnosisData Diagnosis in all codings for one AR	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE00C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one AR	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE00D – 0xE00F	Reserved	—	—	—	—	—	—	—	—

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xE010	DiagnosisData Maintenance required in channel coding for one AR	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE011	DiagnosisData Maintenance demanded in channel coding for one AR	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE012	DiagnosisData Maintenance required in all codings for one AR	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE013	DiagnosisData Maintenance demanded in all codings for one AR	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE014 – 0xE02F	Reserved	—	—	—	—	—	—	—	—
0xE030	PE_EntityFilterData for one AR	Needed	Possible	Read	Diagnostics	—	—	—	Profile
0xE031	PE_EntityStatusData for one AR	Needed	Possible	Read	Diagnostics	—	—	—	Profile
0xE032 – 0xE03F	Reserved	—	—	—	—	—	—	—	—
0xE040	WriteMultiple [a]	—	—	Write	—	—	—	—	—
0xE041	ApplicationReadyBlock [b]	Needed	Possible	Read	Diagnostics	—	—	—	Yes
0xE042 – 0xE04F	Reserved	—	—	—	—	—	—	—	—
0xE050	Reserved (legacy)	—	—	—	—	—	—	—	—
0xE051 – 0xE05F	Reserved for FastStartUp	—	—	—	—	—	—	—	—
0xE060	RS_GetEvent	Needed	Possible	Read	Diagnostics	—	—	—	No
0xE061	RS_AckEvent	Needed	—	Write	—	Controller, Operator-Station	—	—	—
0xE062 – 0xEBFF	Reserved	—	—	—	—	—	—	—	—
0xEC00 – 0xEFFF	Reserved for profiles	Needed	—	—	Diagnostics	—	—	—	—

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
Needed: Reference to an AR needed. Any service without this reference shall return an error.									
[a] The ACD Roles for the contained records apply.									
[b] The reading of this index closes the prm phase, which is opened by Connect, PrmBegin or Plug/Release.									

The coding for API specific defined record data shall be according to Table 570 and the usage of the address parameters according to Table 561.

Table 570 – Index (API specific)

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xF000	RealIdentificationData for one API	—	—	Read	Diagnostics	—	—	—	No
0xF001 – 0xF009	Reserved	—	—	—	—	—	—	—	—
0xF00A	DiagnosisData Diagnosis in channel coding for one API	—	Possible	Read	Diagnostics	—	—	—	No
0xF00B	DiagnosisData Diagnosis in all codings for one API	—	Possible	Read	Diagnostics	—	—	—	No
0xF00C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one API	—	Possible	Read	Diagnostics	—	—	—	No
0xF00D – 0xF00F	Reserved	—	—	—	—	—	—	—	—
0xF010	DiagnosisData Maintenance required in channel coding for one API	—	Possible	Read	Diagnostics	—	—	—	No
0xF011	DiagnosisData Maintenance demanded in channel coding for one API	—	Possible	Read	Diagnostics	—	—	—	No

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xF012	DiagnosisData Maintenance required in all codings for one API	—	Possible	Read	Diagnostics	—	—	—	No
0xF013	DiagnosisData Maintenance demanded in all codings for one API	—	Possible	Read	Diagnostics	—	—	—	No
0xF014 – 0xF01F	Reserved	—	—	—	—	—	—	—	—
0xF020	ARData for one API	—	Possible	Read	Diagnostics	—	—	—	No
0xF021 – 0xF3FF	Reserved	—	—	—	—	—	—	—	—
0xF400 – 0xF7FF	Reserved for profiles	—	—	—	Diagnostics	—	—	—	—

The coding for device specific defined record data shall be according to Table 571 and the usage of the address parameters according to Table 562.

Table 571 – Index (device specific)

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xF800 – 0xF80B	Reserved	—	—	—	—	—	—	—	—
0xF80C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one device	—	Possible	Read	Diagnostics	—	—	—	No
0xF80D – 0xF81F	Reserved	—	—	—	—	—	—	—	—
0xF820	ARData	—	Possible	Read	Diagnostics	—	—	—	No
0xF821	APIData	—	—	Read	Diagnostics	—	—	—	No
0xF822 – 0xF82F	Reserved	—	—	—	—	—	—	—	—
0xF830	LogBookData	—	Possible	Read	Diagnostics	—	—	—	No
0xF831	PdevData	—	Possible	Read	Diagnostics	—	—	—	—

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xF832 – 0xF83F	Reserved	—	—	—	—	—	—	—	—
0xF840	I&M0FilterData	—	—	Read	Diagnostics	—	—	—	No
0xF841	PDRealData	—	—	Read	Diagnostics	—	—	—	No
0xF842	PDExpectedData	—	Possible	Read	Diagnostics	—	—	—	Yes
0xF843 – 0xF84F	Reserved	—	—	—	—	—	—	—	—
0xF850 – 0xF85F	Reserved	—	—	—	—	—	—	—	—
0xF860	GSD upload using UploadBLOBQuery and UploadBLOB	—	—	Read	Diagnostics	—	—	—	No
0xF861 – 0xF86F	Reserved for controller to controller communication	—	—	—	—	—	—	—	—
0xF870	PE_EntityFilterData	—	Possible	Read	Diagnostics	—	—	—	Profile
0xF871	PE_EntityStatusData	—	Possible	Read	Diagnostics	—	—	—	Profile
0xF872 – 0xF87F	Reserved	—	—	—	—	—	—	—	—
0xF880	AssetManagementData Contains all (if only this record is supported) or the first chunk of complete assets (if 0xF881 is supported)	—	Possible	Read	Diagnostics	—	—	—	No
0xF881	AssetManagementData Contains the second chunk of complete assets	—	Possible	Read	Diagnostics	—	—	—	No
0xF882	AssetManagementData Contains the third chunk of complete assets (if 0xF881 is supported)	—	Possible	Read	Diagnostics	—	—	—	No
0xF883	AssetManagementData Contains the fourth chunk of complete assets (if 0xF882 is supported)	—	Possible	Read	Diagnostics	—	—	—	No

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xF884	AssetManagementData Contains the fifth chunk of complete assets (if 0xF883 is supported)	—	Possible	Read	Diagnostics	—	—	—	No
0xF885	AssetManagementData Contains the sixth chunk of complete assets (if 0xF884 is supported)	—	Possible	Read	Diagnostics	—	—	—	No
0xF886	AssetManagementData Contains the seventh chunk of complete assets (if 0xF885 is supported)	—	Possible	Read	Diagnostics	—	—	—	No
0xF887	AssetManagementData Contains the eighth chunk of complete assets (if 0xF886 is supported)	—	Possible	Read	Diagnostics	—	—	—	No
0xF888	AssetManagementData Contains the ninth chunk of complete assets (if 0xF887 is supported)	—	Possible	Read	Diagnostics	—	—	—	No
0xF889	AssetManagementData Contains the tenth chunk of complete assets (if 0xF888 is supported)	—	Possible	Read	Diagnostics	—	—	—	No
0xF88A – 0xF8EF	Reserved	—	—	—	—	—	—	—	—
0xF8F0	Stream Add using UNIAddStreamReq and UNIAddStreamRsp	—	—	Read Query	NetworkManagement	NetworkManagement	—	—	No
0xF8F1	PDRsiInstances [a]	—	—	Read	Diagnostics	—	—	—	No
0xF8F2	Stream Remove using UNIRemoveStreamReq and UNIRemoveStreamRsp	—	Possible	Read Query	NetworkManagement	NetworkManagement	—	—	No
0xF8F3	Stream Renew using UNIRenewStreamReq and UNIRenewStreamRsp	—	Possible	Read Query	NetworkManagement	NetworkManagement	—	—	No
0xF8F4 – 0xF8FF	Reserved	—	—	—	—	—	—	—	—

Value (hexadecimal)	Meaning of index	ARUUID context	NULL response	Access	ACD Role Read	ACD Role Write	Persistence	Startup parameter	SRL Primary AR context
0xF900	CIMSecurityServiceBlock Reserved for security	—	Possible	Read Query	Diagnostics	see 5.2.3.4.4	Yes	—	No
0xF901	CIMDCPService	—	—	Read Query	Diagnostics	Controller, Operator-Station	Yes	—	No
0xF902	CIMNetConfService	—	—	Read Query	Diagnostics	Controller, Operator-Station	Yes	—	No
0xF903 – 0xF91F	Reserved	—	—	—	—	—	—	—	—
0xF920	CIMElectricPowerReal	—	—	Read	Diagnostics	—	—	—	No
0xF921 – 0xFBFE	Reserved	—	—	—	—	—	—	—	—
0xFBFF	Trigger index for the RPC connection monitoring inside the CMSM	—	Possible	Read	Diagnostics	—	—	—	—
0xFC00 – 0xFFFF	Reserved for profiles	—	—	—	Diagnostics	—	—	—	—

[a] This block shall contain only data that matches the responding DLPDU SourceAddress.

5.2.4 Coding section related to AL services

5.2.4.1 Coding of the field RecordDataLength

This field shall be coded as data type Unsigned32 according to Table 572. The field RecordDataLength shall only contain the number of user octets.

Table 572 – RecordDataLength

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	RecordDataRead: Size of the receive buffer at the RPC client. The server shall respond with the real size of the requested object or reject the service with an ErrorCode. RecordDataWrite: Expected size of the receive buffer at the RPC server. The server shall provide this buffer or reject the service with an ErrorCode.

5.2.4.2 Coding of the field SeqNumber

This field shall be coded as data type Unsigned16 and shall be incremented by one for each successful IODWriteReq and IODReadReq by the application of the requestor and mirrored by the responder; the protocol layer shall not check the SeqNumber.

Thus, the check for the correct SeqNumber should be done by the application.

The SeqNumber is valid in the context of a SessionKey and starts with zero.

NOTE The SeqNumber and the SessionKey allow detection of sequence errors.

This field shall contain the value zero for an implicit AR.

5.2.4.3 Coding of the field ARType

This field shall be coded as data type Unsigned16 with the values according to Table 573.

Table 573 – ARType

Value (hexadecimal)	Meaning	Use
0x0000	Reserved	—
0x0001	IOCARSingle	—
0x0002 – 0x0005	Reserved	—
0x0006	IOSAR	The supervisor AR is a special form of the IOCARSingle allowing takeover of the ownership of a submodule
0x0007 – 0x000F	Reserved	—
0x0010	IOCARSingle using RT_CLASS_3	This is a special form of the IOCARSingle indicating RT_CLASS_3 communication
0x0011 – 0x001F	Reserved	—
0x0020	IOCARSR	The SR AR is a special form of the IOCARSingle indicating system redundancy or dynamic reconfiguration usage
0x0021 – 0xFFFF	Reserved	—

5.2.4.4 Coding of the field SessionKey

This field shall be coded as data type Unsigned16 and shall be increased by one for each connect by the CMInitiator. The CMResponder shall use this value for each subsequent PROFINETIOServiceReq- and PROFINETIOServiceResPDU within this session.

NOTE The SessionKey allows the CMInitiator to detect sequence errors during the establishment and release phase of an AR.

This field shall contain the value zero for the implicit AR.

5.2.4.5 Coding of the field CMInitiatorMacAdd

This field shall be coded as data type OctetString[6]. The value of the field CMInitiatorMacAdd shall be according to the 48-bit universal LAN MAC addresses of IEEE Std 802.

5.2.4.6 Coding of the field IOCRMulticastMACAdd

This field shall be coded as data type OctetString[6].

NOTE Octet 1 contains the Individual/Group Address Bit (lsb).

The value of the field IOCRMulticastMACAdd shall be set according to the 48-bit universal LAN MAC addresses of IEEE Std 802, and to Table 574 and Table 575.

The IP Multicast address used for multicast communication relation using RT_CLASS_UDP shall be set according to IETF RFC 2365.

Table 574 – IOCRMulticastMACAdd using RT_CLASS_UDP

IP multicast address	Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Associated FrameID	Usage
239.192.248.0	01-00-5E	40-F8-00	0xF800	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.192.248.1 – 239.192.251.254	01-00-5E	40-F8-01 to 40-FB-FE	0xF801 – 0xFBFE	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.192.251.255	01-00-5E	40-FB-FF	0xFBFF	Used for multicast communication relations in conjunction with RT_CLASS_UDP

Table 575 – IOCRMulticastMACAdd using RT_CLASS_x

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-00 to 00-00-FF	Reserved for other applications
01-0E-CF	00-01-00	Reserved for further multicast addresses within the Type 10 context
01-0E-CF	00-01-01	RT_CLASS_3 destination multicast address
01-0E-CF	00-01-02	RT_CLASS_3 invalid frame multicast address
01-0E-CF	00-01-03 to FF-FF-FF	Reserved for further multicast addresses within the Type 10 context
03-0E-CF	00-00-00 to 00-07-FF	Stream destination multicast address for RT_CLASS_STREAM
03-00-00 to F3-FF-00	00-00-00	Fast Forwarding local administered multicast address range [a]
Other locally administered multicast	—	Reserved for further multicast addresses within the Type 10 context
[a] Fast Forwarding is not usable together with ARProperties.TimeAwareSystem = TimeAware		

The Organizationally Unique Identifier (OUI) as defined by IEEE Std 802 for Type 10 is 00-0E-CF and shall be set according to Table 576.

Table 576 – Type 10 OUI

Value (hexadecimal)	Meaning
00-0E-CF	Global administered individual unicast
01-0E-CF	Global administered group (multicast) address
02-0E-CF	Local administered individual unicast
03-0E-CF	Local administered group (multicast) address

5.2.4.7 Coding of the field CMResponderMacAdd

This field shall be coded as data type OctetString[6]. The value of the field CMResponderMacAdd shall be according to the 48-bit universal LAN MAC addresses of IEEE Std 802.

NOTE Octet 1 contains the Individual/Group Address Bit (lsb).

5.2.4.8 Coding of the field ARProperties

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 2: ARProperties.State
This field shall be set according to Table 577.

Table 577 – ARProperties.State

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Active
0x02 – 0x07	Reserved

Bit 3: ARProperties.SupervisorTakeoverAllowed
This field shall be set according to Table 578.

Table 578 – ARProperties.SupervisorTakeoverAllowed

Value (hexadecimal)	Meaning
0x00	Not allowed
0x01	Allowed

Bit 4: ARProperties.ParameterizationServer
This field shall be set according to Table 579.

Table 579 – ARProperties.ParameterizationServer

Value (hexadecimal)	Meaning
0x00	External PrmServer
0x01	CM Initiator

Bit 5 – 6: ARProperties.ProtectionProperties
This field shall be set to zero. Reserved for security.

Bit 7: ARProperties.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 8: ARProperties.DeviceAccess
This field shall be set according to Table 580.

Table 580 – ARProperties.DeviceAccess

Value (hexadecimal)	Meaning
0x00	Only the submodules from the ExpectedSubmoduleBlock are accessible
0x01	Submodule access is controlled by IO device application This requires: ARType := IOSAR

Bit 9 – 10: ARProperties.CompanionAR
This field shall be set according to Table 581.

Table 581 – ARProperties.CompanionAR

Value (hexadecimal)	Meaning	Use
0x00	Single AR	RT_CLASS_1, RT_CLASS_2, RT_CLASS_3, RT_CLASS_UDP, or RT_CLASS_STREAM connection establishment
0x01	First AR of a companion pair and a companion AR shall follow	StartupMode := Legacy RT_CLASS_3
0x02	Companion AR	StartupMode := Legacy RT_CLASS_3
0x03	Reserved	—

Bit 11: ARProperties.AcknowledgeCompanionAR
This field shall be set according to Table 582.

Table 582 – ARProperties.AcknowledgeCompanionAR

Value (hexadecimal)	Meaning	Use
0x00	No companion AR or no acknowledge for the companion AR required	RT_CLASS_1, RT_CLASS_2, RT_CLASS_3, RT_CLASS_UDP, or RT_CLASS_STREAM connection establishment
0x01	Companion AR with acknowledge	StartupMode := Legacy RT_CLASS_3

Bit 12 – 23: ARProperties.Reserved_2
This field shall be set according to 3.4.2.2.

Bit 24 – 26: ARProperties.Reserved_3
This field shall be set to zero.

Bit 27: ARProperties.RejectDCPsetRequests
This field shall be set according to Table 583.

Table 583 – ARProperties.RejectDCPsetRequests

Value (hexadecimal)	Meaning
0x00	Accepted
0x01	Rejected

Bit 28: ARProperties.TimeAwareSystem
This field shall be set according to Table 584.

Table 584 – ARProperties.TimeAwareSystem

Value (hexadecimal)	Meaning	Use
0x00	NonTimeAware	The AR does not require time-aware system mode. This means: IOCRProperties.RTClass shall be ≠ 5.
0x01	TimeAware	The AR does require time-aware system mode. This means: ARProperties.CombinedObjectContainer shall be set to “COC”. ARProperties.StartupMode shall be set to “Advanced”. IOCRProperties.RTClass shall be set to 5.

Bit 29: ARProperties.CombinedObjectContainer
This field shall be set according to Table 585.

Table 585 – ARProperties.CombinedObjectContainer

Value (hexadecimal)	Meaning	Use
0x00	NoCOC	CombinedObjectContainer not used
0x01	COC	Usage of CombinedObjectContainer required

Bit 30: ARProperties.StartupMode
This field shall be set according to Table 586.

Table 586 – ARProperties.StartupMode

Value (hexadecimal)	Meaning	Use
0x00	Legacy	Recommended for IO Controller and IO Supervisor See IEC 61158-6-10:2010 and Annex B
0x01	Advanced	Mandatory

Bit 31: ARProperties.PullModuleAlarmAllowed
This field shall be set according to Table 587.

Table 587 – ARProperties.PullModuleAlarmAllowed

Value (hexadecimal)	Meaning	Use
0x00	The AlarmType(=Pull) shall signal pulling of submodule and module. The subslot number zero shall indicate pulling of module in conjunction with AlarmType(=Pull).	Mandatory
0x01	The AlarmType(=Pull) shall signal pulling of submodule. The AlarmType(=Pull module) shall signal pulling of module. The subslot number 0 – 0x9FFF shall indicate pulling of submodule in conjunction with AlarmType(=Pull).	Optional

5.2.4.9 Coding of the field IOCRProperties

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 3: IOCRProperties.RTClass
This field shall be set according to Table 588.

Table 588 – IOCRProperties.RTClass

Value (hexadecimal)	Meaning	Use
0x00	Reserved	—
0x01	Reserved (legacy)	—
0x02	Non-time-aware system: RT_CLASS_1 Time-aware system: RT_CLASS_STREAM with “Stream, class RT”	Data-RTC-PDU Use FrameID range 6 for the IOCRs See Table 51
0x03	RT_CLASS_3	Data-RTC-PDU Use FrameID range 3a for the IOCRs See Table 47
0x04	RT_CLASS_UDP	UDP-RTC-PDU Use FrameID range 7 for the IOCRs See Table 52
0x05	Time-aware system: RT_CLASS_STREAM with “Time-aware stream, class HIGH” or “Time-aware stream, class LOW”	Data-RTC-PDU Use FrameID range 3b for the IOCRs See Table 48
0x06 – 0x07	Reserved	—

Bit 4 – 12: IOCRProperties.Reserved_1
This field shall be set to zero.

Bit 13 – 23: IOCRProperties.Reserved_2
This field shall be set according to 3.4.2.2.

Bit 24 – 31: IOCRProperties.Reserved_3
This field shall be set to zero.

5.2.4.10 Coding of the field NumberOfIODataObjects

This field shall be coded as data type Unsigned16.

5.2.4.11 Coding of the field NumberOfIOCS

This field shall be coded as data type Unsigned16.

5.2.4.12 Coding of the field IOCRReference

This field shall be coded as data type Unsigned16. It is an identification tag for the CR and is used within the IOCRBlockReq and IOCRBlockRes to reference the DataItem. Furthermore, it is used in PDIRData of the Physical Device ASE.

5.2.4.13 Coding of the field IOCRTagHeader

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 11: IOCRTagHeader.IOCRVLANID
This field shall be set according to Table 589.

Table 589 – IOCRTagHeader.IOCRVLANID

Value (hexadecimal)	Meaning
0x000	Default NME / CNC defined VLAN according to the requested RT_CLASS. “Non-stream VID” for Stream, class RT, “Stream High VID” or “Stream High Redundant VID” for “Time-aware stream, class HIGH”. “Stream Low VID” or “Stream Low Redundant VID” for “Time-aware stream, class LOW”, and equivalent to “Non-stream VID” for RT_CLASS_1/_2/_3/_UDP for non-time-aware systems.
0x001 – 0xFFF	Reserved

Bit 12: IOCRTagHeader.Reserved
This field shall be set to zero.

Bit 13 – 15: IOCRTagHeader.IOUserPriority
This field shall be set according to Table 590.

Table 590 – IOCRTagHeader.IOUserPriority

Value (hexadecimal)	IO CR Priority	Meaning
Other	—	Reserved
0x03	Stream, class RT	“Stream, class RT” for time-aware systems
0x05	Time-aware stream, class LOW	“Time-aware stream, class LOW” for time-aware systems
0x06	Time-aware stream, class HIGH	RT_CLASS_1/_2/_3/_UDP for non-time-aware systems and “Time-aware stream, class HIGH” for time-aware systems

5.2.4.14 Coding of the field IOCRType

This field shall be coded as data type Unsigned16 with the values according to Table 591.

Table 591 – IOCRType

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Input CR
0x0002	Output CR
0x0003	Multicast Provider CR
0x0004	Multicast Consumer CR
0x0005 – 0xFFFF	Reserved

5.2.4.15 Coding of the field CMInitiatorActivityTimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 592 and Table 593.

Table 592 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess == 0

Value (decimal)	Meaning	Use
0	Reserved	—
1 – 1 000	Allowed values	The IO device monitors the time between Connect response and the subsequent first activity of the IO controller
1 001 – 65 535	Reserved	—

Table 593 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess == 1 or ARProperties.StartupMode == Advanced

Value (decimal)	Meaning	Use
0 – 99	Reserved	—
100 – 1 000	Allowed values	Default value: 200 The IO device monitors the time between Connect response and the subsequent Read and Write record activity of the IO controller
1 001 – 65 535	Reserved	—

This field shall be used by the CMDEV for the calculation of the CMInitiatorActivityTimeout according to Formula (54).

$$\text{CMInitiatorActivityTimeout} = \text{CMInitiatorActivityTimeoutFactor} \times 100\ \text{ms} \quad (54)$$

where

CMInitiatorActivityTimeout is the CM initiator activity timeout

CMInitiatorActivityTimeoutFactor is the CM initiator activity timeout factor

This field shall be used by the CMCTL for the calculation of the CMInitiatorTriggerTimeoutFactor which is used for the AR establishing with the values according to Formula (55) and Table 594.

$$\text{CMInitiatorTriggerTimeoutFactor} = \text{CMInitiatorActivityTimeoutFactor } \mathit{DIV}\ 3 \quad (55)$$

where

CMInitiatorActivityTimeoutFactor is the CM initiator activity timeout factor

CMInitiatorTriggerTimeoutFactor is the CM initiator trigger timeout factor

Table 594 – CMInitiatorTriggerTimeoutFactor

Value (decimal)	Meaning	Use
0 – 32	Reserved	—
33 – 333	With a time base of 100 ms	Used for the connection monitoring during startup until the PPM / CPM connection monitoring takes place.
334 – 65 535	Reserved	—

The value of the CMInitiatorTriggerTimeout shall be calculated according to Formula (56).

$$\text{CMInitiatorTriggerTimeout} = \text{CMInitiatorTriggerTimeoutFactor} \times 100\ \text{ms} \quad (56)$$

where

CMInitiatorTriggerTimeout is the CM initiator trigger timeout

CMInitiatorTriggerTimeoutFactor is the CM initiator trigger timeout factor

5.2.4.16 Coding of the field StationNameLength

This field shall be coded as data type Unsigned16.

5.2.4.17 Coding of the field CMInitiatorStationName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.16.

5.2.4.18 Coding of the field CMResponderStationName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.16.

5.2.4.19 Coding of the field ParameterServerStationName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.16.

5.2.4.20 Coding of the field ProviderStationName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.16.

5.2.4.21 Coding of the field IODataObjectFrameOffset

This field shall be coded as data type Unsigned16 according to Table 595. It is used within the IOCRBlockReq to reference the offset of the DataItem. The number of used octets shall not exceed the DataLength.

The size of a DataItem of a submodule with SubmoduleProperties.Type == NO_IO and SubmoduleProperties.DiscardIOXS == TRUE is zero, thus the offset is allowed to be DataLength.

Table 595 – IODataObjectFrameOffset

Value (hexadecimal)	Meaning
0x0000 – 0x059F	Offset of a DataItem
0x05A0	Offset of a DataItem if SubmoduleProperties.DiscardIOXS == TRUE for this submodule is set.
0x05A1 – 0xFFFF	Reserved

5.2.4.22 Coding of the field IOCSFrameOffset

This field shall be coded as data type Unsigned16 according to Table 596.

It is used within the IOCRBlockReq to reference the offset of the IOCS. The number of used octets shall not exceed the DataLength.

The size of an IOCS of a submodule with SubmoduleProperties.Type == NO_IO and SubmoduleProperties.DiscardIOXS == TRUE is zero, thus the offset is allowed to be DataLength.

Table 596 – IOCSFrameOffset

Value (hexadecimal)	Meaning
0x0000 – 0x059F	Offset of a IOCS
0x05A0	Offset of a IOCS if SubmoduleProperties.DiscardIOXS == TRUE for this submodule is set.
0x05A1 – 0xFFFF	Reserved

5.2.4.23 Coding of the field LengthIOCS

This field shall be coded as data type Unsigned8 and shall be set according to Table 597.

Table 597 – LengthIOCS

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	One octet for IOCS
0x02 – 0xFF	Reserved

5.2.4.24 Coding of the field LengthIOPS

This field shall be coded as data type Unsigned8 and shall be set according to Table 598.

Table 598 – LengthIOPS

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	One octet for IOPS
0x02 – 0xFF	Reserved

5.2.4.25 Coding of the field LengthData

This field shall be coded as data type Unsigned16 according to Table 599.

Table 599 – LengthData

Value (hexadecimal)	Meaning
0x0000 – 0x059F	Length of data
0x05A0 – 0xFFFF	Reserved

5.2.4.26 Coding of the field NumberOfAPIs

This field shall be coded as data type Unsigned16.

NOTE The value 0 is only used in conjunction with ARProperties.DeviceAccess = 1.

5.2.4.27 Coding of the field AlarmCRProperties

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0: AlarmCRProperties.Priority
This field shall be set according to Table 600.

Table 600 – AlarmCRProperties.Priority

Value (hexadecimal)	Meaning
0x00	User priority (default) The priority given by the user is used and two alarm resources are available
0x01	Fixed priority Use only low priority, user priority is ignored and only one alarm resource is available

An IO device as alarm sink shall always support AlarmCRProperties.Priority == ”User priority” and the IO controller as alarm source may choose whether it limits itself to one resource or not.

Bit 1: AlarmCRProperties.Transport
This field shall be set according to Table 601.

Table 601 – AlarmCRProperties.Transport

Value (hexadecimal)	Meaning	Usage
0x00	RTA_CLASS_1	Alarm CR uses DATA-RTA-PDU
0x01	RTA_CLASS_UDP	Alarm CR uses UDP-RTA-PDU

Bit 2 – 23: AlarmCRProperties.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 24 – 31: AlarmCRProperties.Reserved_2
This field shall be set to zero.

5.2.4.28 Coding of the field AlarmCRTagHeaderHigh

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 11: AlarmCRTagHeaderHigh.AlarmCRVLANID
This field shall be set according to Table 602.

Table 602 – AlarmCRTagHeaderHigh.AlarmCRVLANID

Value (hexadecimal)	Meaning
0x000	Default NME / CNC defined VLAN for the selected AlarmUserPriority. “Non-stream VID” for RTA_CLASS_1/_UDP for time-aware systems, and equivalent to “Non-stream VID” for RTA_CLASS_1/_UDP for non-time-aware systems.
0x001 – 0xFFF	Reserved

Bit 12: AlarmCRTagHeaderHigh.Reserved
This field shall be set to zero.

Bit 13 – 15: AlarmCRTagHeaderHigh.AlarmUserPriority
This field shall be set according to Table 603.

Table 603 – AlarmCRTagHeaderHigh.AlarmUserPriority

Value (hexadecimal)	Meaning	Comments
0x00 – 0x05	Reserved	—
0x06	Alarm CR Priority High	The content field is independent from the used TCI.PCP value. The assigned queue is defined by the end station interface configuration.
0x07	Reserved	—

5.2.4.29 Coding of the field AlarmCRTagHeaderLow

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 11: AlarmCRTagHeaderLow.AlarmCRVLANID
This field shall be set according to Table 604.

Table 604 – AlarmCRTagHeaderLow.AlarmCRVLANID

Value (hexadecimal)	Meaning
0x000	Default NME / CNC defined VLAN for the selected AlarmUserPriority. “Non-stream VID” for RTA_CLASS_1/_UDP for time-aware systems, and equivalent to “Non-stream VID” for RTA_CLASS_1/_UDP for non-time-aware systems.
0x001 – 0xFFF	Reserved

Bit 12: AlarmCRTagHeaderLow.Reserved
This field shall be set to zero.

Bit 13 – 15: AlarmCRTagHeaderLow.AlarmUserPriority
This field shall be set according to Table 605.

Table 605 – AlarmCRTagHeaderLow.AlarmUserPriority

Value (hexadecimal)	Meaning	Comments
0x00 – 0x04	Reserved	—
0x05	Alarm CR Priority Low	The content field is independent from the used TCI.PCP value. The assigned queue is defined by the end station interface configuration.
0x06 – 0x07	Reserved	—

5.2.4.30 Coding of the field AlarmSequenceNumber

This field shall be coded as data type Unsigned16 with the values according to Table 606.

Table 606 – AlarmSequenceNumber

Value (hexadecimal)	Meaning	Use
0x0000 – 0x07FF	Mirrors the sequence number of the plug alarm notification	It provides the relation between Plug Alarm notification and the ControlBlockPlug within Application Ready
0x0800 – 0xFFFF	Reserved	—

5.2.4.31 Coding of the field AlarmCRType

This field shall be coded as data type Unsigned16 with the values according to Table 607.

Table 607 – AlarmCRType

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Alarm CR
0x0002 – 0xFFFF	Reserved

5.2.4.32 Coding of the field RTATimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 608. The time base shall be 100 ms.

The RTATimeout shall be calculated according to Formula (57).

$$RTATimeout = RTATimeoutFactor \times 100\ ms \quad (57)$$

where

RTATimeout is the RTA timeout

RTATimeoutFactor is the RTA timeout factor

Table 608 – RTATimeoutFactor

Value (hexadecimal)	Meaning	Use
0x0000	Reserved	—
0x0001	Mandatory	Default DATA-RTA-PDU or UDP-RTA-PDU acknowledge monitoring
0x0002 – 0x0064	Mandatory	DATA-RTA-PDU or UDP-RTA-PDU acknowledge monitoring
0x0065 – 0xFFFF	Optional	DATA-RTA-PDU or UDP-RTA-PDU acknowledge monitoring

NOTE The value of the RTATimeoutFactor depends on the conveyance time of the DATA-RTA-PDU or UDP-RTA-PDU between two peers.

5.2.4.33 Coding of the field RTARetries

This field shall be coded as data type Unsigned16 with the values according to Table 609.

Table 609 – RTARetries

Value (hexadecimal)	Meaning	Use
0x0000 – 0x0002	Reserved	Reserved
0x0003	Mandatory	Default
0x0004 – 0x000F	Mandatory	—
0x0010 – 0xFFFF	Reserved	Reserved

5.2.4.34 Coding of the field PROFINETIOConstantValue

This field shall be coded as data type Structure with values according to Table 610.

Table 610 – PROFINETIOConstantValue

Name	Coding	Value (hexadecimal)
PROFINETIOConstantValue.Data1	Unsigned32	See Table 611
PROFINETIOConstantValue.Data2	Unsigned16	0x6C97
PROFINETIOConstantValue.Data3	Unsigned16	0x11D1
PROFINETIOConstantValue.Data4	Unsigned8, Unsigned8	0x82, 0x71

NOTE The ordering of transmission of the Unsigned32 or Unsigned16 values is according to the RPC Flag Drep (little endian or big endian) if it is part of the RPCHeader or NDREPMapPDU. If it is part of the NDRDataXxx PDUs or RTA-PDU then only big endian format is used.

Table 611 – PROFINETIOConstantValue.Data1

Value (hexadecimal)	Meaning	Use
0xDEA00000	RPC	The value of the field PROFINETIOConstantValue shall be DEA00000-6C97-11D1-8271.
0xDEA00001	RSI	The value of the field PROFINETIOConstantValue shall be DEA00001-6C97-11D1-8271. This value shall only be used in conjunction with ARBlockReq.ARProperties.StartupMode == Advanced. NOTE Used only by CMInitiatorObjectUUID.

5.2.4.35 Coding of the field AddressResolutionProperties

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 2: AddressResolutionProperties.Protocol
This field shall be set according to Table 612.

Table 612 – AddressResolutionProperties.Protocol

Value (hexadecimal)	Meaning	Use
0x00	Reserved	—
0x01	DNS	Multicast consumer uses DNS to resolve the multicast provider source MAC address.
0x02	DCP	Multicast consumer uses DCP to resolve the multicast provider source MAC address.
0x03 – 0x07	Reserved	—

Bit 3 – 7: AddressResolutionProperties.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 8 – 15: AddressResolutionProperties.Reserved_2
This field shall be set to zero.

Bit 16 – 31: AddressResolutionProperties.Factor
This field shall be coded with the values according to Table 613. The time base shall be one second. The AddressResolutionInterval shall be calculated according to Formula (58).

Table 613 – AddressResolutionProperties.Factor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x0064	Mandatory
0x0065 – 0xFFFF	Optional

$$AddressResolutionInterval = AddressResolutionProperties.Factor \times 1\ s \quad (58)$$

where

AddressResolutionInterval is the address resolution interval

AddressResolutionProperties.Factor is the factor for the calculation of the address resolution interval

5.2.4.36 Coding of the field MCITimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 614. The time base shall be 100 ms. The MCIMonitoringInterval shall be calculated according to Formula (59).

Table 614 – MCITimeoutFactor

Value (hexadecimal)	Meaning
0x0000 – 0x0064	Mandatory
0x0065 – 0xFFFF	Reserved

$$MCIMonitoringInterval = MCITimeoutFactor \times 100\ ms \quad (59)$$

where

MCIMonitoringInterval is the MCI monitoring interval

MCITimeoutFactor is the factor for the calculation of the MCI monitoring interval

5.2.4.37 Coding of fields related to Instance, DeviceID, VendorID

5.2.4.37.1 General

The Unsigned16 fields Instance, DeviceID and VendorID are divided into two Unsigned8 fields to define the order of the octets inside the ObjectUUID OctetString.

5.2.4.37.2 Coding of the field InstanceLow

This field, defined as an administrative number, shall be coded as data type Unsigned8 according to Table 615 and Table 616.

Table 615 – InstanceLow and InstanceHigh

InstanceID		Meaning
InstanceHigh Value (hexadecimal)	InstanceLow Value (hexadecimal)	
Bit 3 to Bit 0	Bit 7 to Bit 0	
0x0000		The value zero for both InstanceHigh and InstanceLow is recommended for devices implementing only a single instance.
0x0001 – 0x0FFF		Manufacturer specific identification of an instance implemented in a device. A device may implement multiple instances of IO controller and/or IO device.

Table 616 – InstanceHigh

InstanceHigh Value (hexadecimal)	Meaning
Bit 7 to Bit 4	
0x00	Interface 1 (Default) Identifies the end station component of a device assigned to this instance. See “I” within Formula (53)
0x01 – 0x0E	—
0x0F	Interface 16

5.2.4.37.3 Coding of the field InstanceHigh

This field, defined as an administrative number, shall be coded as data type Unsigned8 according to Table 615 and Table 616.

5.2.4.37.4 Coding of the field DeviceIDLow

This field, defined as an administrative number, shall be coded as data type Unsigned8 according to Table 617.

Table 617 – DeviceIDLow and DeviceIDHigh

DeviceID		Meaning
DeviceIDHigh Value (hexadecimal)	DeviceIDLow Value (hexadecimal)	
0x0000		The value zero for both DeviceIDHigh and DeviceIDLow is reserved.
0x0001 – 0xFFFF		Manufacturer specific

NOTE VendorID, DeviceID, OrderID, HWRevision and IM_Serial_Number uniquely identifies one particular hardware.

5.2.4.37.5 Coding of the field DeviceIDHigh

This field, defined as an administrative number, shall be coded as data type Unsigned8 according to Table 617.

5.2.4.37.6 Coding of the field VendorIDLow

This field, defined as an administrative number, shall be coded as data type Unsigned8 with the values according to Table 618.

Table 618 – VendorIDLow and VendorIDHigh

VendorID		Meaning
VendorIDHigh Value (hexadecimal)	VendorIDLow Value (hexadecimal)	
0x0000		The value zero for both VendorIDHigh and VendorIDLow is reserved for IEC 61158-6-3 devices.
0x0001 – 0xFEFF		Administrative number identifying the manufacturer.
0xF000 – 0xF0FF		Reserved for profiles
0xF100		Used for Process Automation profile devices
0xF101 – 0xFFFE		Reserved for profiles
0xFFFF		Used for IEC 61131-9 profile devices

5.2.4.37.7 Coding of the field VendorIDHigh

This field, defined as an administrative number, shall be coded as data type Unsigned8 with the values according to Table 618.

5.2.4.38 Coding of the field ModuleIdentNumber

This field shall be coded as data type Unsigned32. This field shall be coded with the values according to Table 619.

Table 619 – ModuleIdentNumber

Value (hexadecimal)	Meaning
0x00000000	Used in conjunction with – ModuleDiffBlock together with ModuleState := NoModule Otherwise reserved
0x00000001 – 0xFFFFFFFF	Manufacturer specific

5.2.4.39 Coding of the field SubmoduleIdentNumber

This field shall be coded as data type Unsigned32. This field shall be coded with the values according to Table 620.

Table 620 – SubmoduleIdentNumber

Value (hexadecimal)	Meaning
0x00000000	Used in conjunction with – ModuleDiffBlock together with SubmoduleState.IdentInfo := NoSubmodule – functionality pull module together with SubslotNumber := 0 and ARProperties.PullModuleAlarmAllowed := 0 Should not be used manufacturer specific
0x00000001 – 0xFFFFFFFF	Manufacturer specific

The numbering for device identification is hierarchical. The toplevel is the DeviceIdentNumber which contains an uniquely assigned administrative number. The second level is the ModuleIdentNumber that is unique in the scope of the DeviceIdentNumber. The third level is the SubmoduleIdentNumber that is unique in the scope of the ModuleIdentNumber.

NOTE 1 Unique means, that the associated submodule properties, for example data structure, diagnosis and parameterization, defined in this document are identified.

NOTE 2 The hardware of a submodule is identified by the VendorID and the OrderID.

5.2.4.40 Coding of the field NumberOfARs

This field shall be coded as data type Unsigned16.

5.2.4.41 Coding of the field NumberOfIOCRs

This field shall be coded as data type Unsigned16.

5.2.4.42 Coding of the field SubmoduleDataLength

This field shall be coded as data type Unsigned16. The range shall be between 0 and 1 439.

5.2.4.43 Coding of the field NumberOfModules

This field shall be coded as data type Unsigned16.

5.2.4.44 Coding of the field NumberOfSlots

This field shall be coded as data type Unsigned16.

5.2.4.45 Coding of the field NumberOfSubmodules

This field shall be coded as data type Unsigned16.

5.2.4.46 Coding of the field NumberOfSubslots

This field shall be coded as data type Unsigned16.

5.2.4.47 Coding of the field ARUUID

This field shall be coded as data type UUID according to Table 621, Table 622, Table 623, and Table 624.

Table 621 – ARUUID

Value (UUID)	Meaning	Use
00000000-0000-0000-0000-000000000000	Reserved	The value NIL indicates the usage of the implicit AR.
Other	Valid for an AR	Unambiguous value which identifies an AR. See Table 622 for IOCARSR

Table 622 – ARUUID in conjunction with ARType==IOCARSR

Value	Description		
aaaaaaaa-bbbb-cccc-dddd-eeeeeeeezzzz	Identifies an application relation where		
	aaaaaaaa bbbb (6 octets)	Discriminator Is similar to the standard ARUUID	
	cccc dddd eeeeeeee (8 octets)	ConfigID This part is used as a configuration info for dynamic reconfiguration covering the content of the IODConnect.req.	
	zzzz (2 octets)	Selector This part is used to select between the ARs of an ARset. This IOCARSR differs only in Bit 0 – 2.	
		Bit 0 – 2:	Arnumber 01: 1st AR of an ARset 02: 2nd AR of an ARset 03: 3rd AR of an ARset 04: 4th AR of an ARset Others: Reserved
		Bit 3 – 4:	Arresource 02: Communication endpoint shall allocate two ARs for the ARset Others: Reserved
		Bit 5 – 15:	Reserved

Table 623 – Conjunction between ARUUID.Arnumber and Endpoint1 or Endpoint2

ARUUID.Arnumber	Endpoint	Usage
1	Left or above	Shall be used if only one endpoint exists; and for Endpoint1
2		
3	Right or below	Shall be used for Endpoint2
4		

The ARUUID.ConfigID shall be created as shown in Table 624. An MD5 hash, calculated from the content of the IODConnect.req without the IO controller dependent parts, may be used.

Table 624 – ARUUID.ConfigID generation rule

Content of the IODConnect.req without the IO controller dependent parts as	
within ARBlockReq	the elements ARUUID, SessionKey, CMInitiatorMacAdd, CMInitiatorObjectUUID, InitiatorUDPRTPort, StationNameLength, CMInitiatorStationName,
within IOCRBlockReq	the elements IOCRReference, FrameID, Phase, Sequence, FrameSendOffset, IOCRMulticastMACAdd,
within AlarmCRBlockReq	the element LocalAlarmReference, and
within ARRPCBlockReq	the element InitiatorRPCServerPort.

5.2.4.48 Coding of the field TargetARUUID

This field shall be coded as data type UUID according to Table 625.

Table 625 – TargetARUUID

Value (UUID)	Meaning	Use
00000000-0000-0000-0000-000000000000	Reserved	Reserved
Other	Valid for an AR	Used to identify an established AR which is the target of the service.

5.2.4.49 Coding of the field AdditionalValue1 and AdditionalValue2

This field shall be coded as data type Unsigned16 according to Table 626.

Table 626 – AdditionalValue1 and AdditionalValue2

Value (hexadecimal)	Meaning	Use
0x0000	No further information or positive response	IODReadRes, IODWriteRes, IODWriteMultipleRes
Other	Additional user information within negative response	IODReadRes, IODWriteRes, IODWriteMultipleRes

5.2.4.50 Coding of the field ControlBlockProperties

The coding of this field shall be according to 3.4.2.3.4, Table 627, and Table 628.

Table 627 – ControlBlockProperties in conjunction with ControlCommand.ApplicationReady

Bitposition	Value (hexadecimal)	Meaning
Bit 0 – 15	—	Shall be set according to 3.4.2.2.

Table 628 – ControlBlockProperties in conjunction with the other values of the field ControlCommand

Bitposition	Value (hexadecimal)	Meaning
Bit 0 – 15	—	Shall be set according to 3.4.2.2.

5.2.4.51 Coding of the field ControlCommand

The coding of this field shall be according to 3.4.2.3.4. Only one of the following bits shall be set. The individual bits shall have the following meaning:

Bit 0: ControlCommand.PrmEnd
This field shall be set according to Table 629.

Table 629 – ControlCommand.PrmEnd

Value (hexadecimal)	Meaning	Use
0x00	No PrmEnd	—
0x01	The IO controller has finished the transmission of its stored startup parameter.	IODControlReq

Bit 1: ControlCommand.ApplicationReady
This field shall be set according to Table 630.

Table 630 – ControlCommand.ApplicationReady

Value (hexadecimal)	Meaning	Use
0x00	No Application Ready	—
0x01	The IO device has finished the startup of its application or a new module or submodule was plugged and is ready to operate.	IOXControlReq

Bit 2: ControlCommand.Release
This field shall be set according to Table 631.

Table 631 – ControlCommand.Release

Value (hexadecimal)	Meaning	Use
0x00	No Release	—
0x01	The IO controller terminates the AR.	IODReleaseReq

Bit 3: ControlCommand.Done
This field shall be set according to Table 632.

Table 632 – ControlCommand.Done

Value (hexadecimal)	Meaning	Use
0x00	No Done	—
0x01	Acknowledge is sent.	IODReleaseRes, IOXControlRes, IODControlRes

Bit 4: ControlCommand.ReadyForCompanion
This field shall be set according to Table 633.

Table 633 – ControlCommand.ReadyForCompanion

Value (hexadecimal)	Meaning	Use
0x00	Not ready for companion	—
0x01	The IO device has finished the startup of its application and is ready for the companion AR.	IOXControlReq

Bit 5: ControlCommand.ReadyForRT_CLASS_3
This field shall be set according to Table 634.

Table 634 – ControlCommand.ReadyForRT_CLASS_3

Value (hexadecimal)	Meaning	Use
0x00	Not ready for RT_CLASS_3	—
0x01	The IO device has finished the startup of its application and is ready for RT_CLASS_3 communication.	IOXControlReq

Bit 6: ControlCommand.PrmBegin
This field shall be set according to Table 635.

Table 635 – ControlCommand.PrmBegin

Value (hexadecimal)	Meaning	Use
0x00	No PrmBegin	—
0x01	The IO controller starts the transmission of its stored startup parameter.	IODControlReq

Bit 7 to 15: ControlCommand.Reserved
This field shall be set to zero.

5.2.4.52 Coding of the field DataDescription

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: DataDescription.Type
This field shall be set according to Table 636.

Table 636 – DataDescription.Type

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Input
0x02	Output
0x03	Reserved

Bit 2 – 15: DataDescription.Reserved
This field shall be set to zero.

5.2.4.53 Coding of the field DataLength

This field shall be coded as data type Unsigned16. The values shall contain the length of the C_SDU or the CSF_SDU according to Table 637.

NOTE The field DataLength is the sum of all DataItems as a minimum. It can be larger.

Table 637 – Values of DataLength

Value (decimal)	Domain	Meaning
40 – 1 440	RT_CLASS_1, RT_CLASS_2, RT_CLASS_3 and RT_CLASS_STREAM	Valid range
12 – 1 440	RT_CLASS_UDP	Valid range
Other	—	Reserved

5.2.4.54 Coding of the field GAP

This field shall be coded as data type Unsigned8. The value shall be set to zero and should not be checked by the receiver.

5.2.4.55 Coding of the field Padding

This field shall be coded as data type Unsigned8. The value shall be set to zero and should not be checked by the receiver.

5.2.4.56 Coding of the field RTCPadding

This field shall be coded as data type Unsigned8.

5.2.4.57 Coding of the field RWPadding

This field shall be coded as data type Unsigned8. The value shall be set to zero.

5.2.4.58 Coding of the field SendClockFactor

This field shall be coded as data type Unsigned16 with values according to Table 638 and Table 639. The time base shall be 31,25 µs.

Table 638 – Values of SendClockFactor with time-base 31,25 µs

Value (decimal)	LengthOfPeriod	< 100 Mbps	100 Mbps	> 100 Mbps
0	—	Reserved	Reserved	Reserved
1	31,25 µs	Reserved	Optional	Optional
2	62,5 µs	Reserved	Optional	Optional
3	93,75 µs	Reserved	Optional	Optional
4	125 µs	Reserved	Optional	Optional
5 – 7	—	Reserved	Optional	Optional
8	250 µs	Reserved	Optional	Optional
9 – 15	—	Reserved	Optional	Optional
16	500 µs	Reserved	Optional	Optional
17 – 31	—	Reserved	Optional	Optional
32	1 ms	Mandatory	Mandatory	Mandatory

Value (decimal)	LengthOfPeriod	< 100 Mbps	100 Mbps	> 100 Mbps
33 – 63	—	Reserved	Optional	Reserved
64	2 ms	Reserved	Optional	Reserved
65 – 127	—	Reserved	Optional	Reserved
128	4 ms	Reserved	Optional	Reserved
129 – 65 535	—	Reserved	Reserved	Reserved

The LengthOfPeriod corresponds with a value range from 31,25 µs to 4 000 µs.

Table 639 – Values of SendClockFactor with time-base 25 µs

Value (decimal)	LengthOfPeriod	< 100 Mbps	100 Mbps	> 100 Mbps
0 – 1 000	—	Reserved	Reserved	Reserved
1 001	25 µs	Reserved	Optional	Optional
1 002	50 µs	Reserved	Optional	Optional
1 003	75 µs	Reserved	Optional	Optional
1 004	100 µs	Reserved	Optional	Optional
1 005 – 1 007	—	Reserved	Optional	Optional
1 008	200 µs	Reserved	Optional	Optional
1 009	—	Reserved	Optional	Optional
1 010	250 µs	Reserved	Optional	Optional
1 011 – 1 015	—	Reserved	Optional	Optional
1 0016	400 µs	Reserved	Optional	Optional
1 017 – 1 019	—	Reserved	Optional	Optional
1 020	500 µs	Reserved	Optional	Optional
1 021 – 1 039	—	Reserved	Optional	Optional
1 040	1 ms	Optional	Optional	Optional
1 041 – 65 535	—	Reserved	Reserved	Reserved

The LengthOfPeriod corresponds with a value range from 25 µs to 1 000 µs.

A maximum Ethernet frame contains 1 522 octets, if envelope frames are supported 2 000 octets. To transfer this kind of frame the limitation from Table 640 applies.

Table 640 – Frame size vs. SendClockFactor

Data rate [Mbit/s]	Minimum LengthOf-Period [1 522 octets]	Minimum LengthOf-Period [2 000 octets]
10	1 217,6 µs	1 600 µs
100	121,76 µs	160 µs
1 000	12,176 µs	16 µs
10 000	1,2176 µs	1,6 µs

The value of the LengthOfPeriod with time-base 31,25 µs shall be calculated according to Formula (60).

$$\text{LengthOfPeriod} = \text{SendClockFactor} \times 31{,}25\ \mu s \quad (60)$$

where

LengthOfPeriod is the length of the period

SendClockFactor is the factor of the send clock, the length of one phase

Used within the PDSyncData block, this field shall only be valid if the field SyncProperties.SyncID contains the value 0.

The value of the LengthOfPeriod with time-base 25 µs shall be calculated according to Formula (61).

$$\text{LengthOfPeriod} = (\text{SendClockFactor} - 1000) \times 25\ \mu s \quad (61)$$

where

LengthOfPeriod is the length of the period

SendClockFactor is the factor of the send clock, the length of one phase

5.2.4.59 Coding of the field ReductionRatio

This field shall be coded as data type Unsigned16 with the values according to Table 641, Table 642, Table 643, Table 644, and Table 645.

The allowed value shall be calculated in conjunction with Table 638 according to Formula (62).

Table 641 – Values of ReductionRatio for RT_CLASS_1, RT_CLASS_2, and RT_CLASS_STREAM

Value (decimal)	< 100 Mbps	100 Mbps	> 100 Mbps	Use
1	Reserved	Mandatory	Mandatory	Transmit and receive every send clock
2	Reserved	Mandatory	Mandatory	Transmit and receive every second send clock
4	Reserved	Mandatory	Mandatory	Transmit and receive every fourth send clock
8	Mandatory	Mandatory	Mandatory	Transmit and receive every eighth send clock
16	Mandatory	Mandatory	Mandatory	Transmit and receive every 16th send clock
32	Mandatory	Mandatory	Mandatory	Transmit and receive every 32th send clock
64	Mandatory	Mandatory	Mandatory	Transmit and receive every 64th send clock
128	Mandatory	Mandatory	Mandatory	Transmit and receive every 128th send clock
256	Mandatory	Mandatory	Mandatory	Transmit and receive every 256th send clock
512	Mandatory	Mandatory	Mandatory	Transmit and receive every 512th send clock
513 – 65 535	Reserved	Reserved	Reserved	—
Other	Optional	Optional	Optional	Not intended to be used

Table 642 – Values of ReductionRatio for RT_CLASS_3 and SendClockFactor ≥ 8

Value (decimal)	Meaning	Use
1	Mandatory	Transmit and receive every send clock
2	Mandatory	Transmit and receive every second send clock
4	Mandatory	Transmit and receive every fourth send clock
8	Mandatory	Transmit and receive every eighth send clock
16	Mandatory	Transmit and receive every sixteenth send clock
17 – 65 535	Reserved	—
Other	Optional	Not intended to be used

Table 643 – Values of ReductionRatio for RT_CLASS_3 and SendClockFactor < 8

Value (decimal)	Meaning	Use
1	Mandatory	Transmit and receive every send clock
2 – 65 535	Reserved	—
Other	Optional	Not intended to be used

Table 644 – Values of ReductionRatio in conjunction with a non-power of 2 SendClockFactor

Value (decimal)	Meaning	Use
1	Optional	Transmit and receive every send clock
2 – 65 535	Reserved	—
Other	Optional	Not intended to be used

Table 645 – Values of ReductionRatio for RT_CLASS_UDP

Value (decimal)	Meaning	Use
1	Optional	Transmit and receive every send clock
2	Optional	Transmit and receive every second send clock
4	Optional	Transmit and receive every fourth send clock
8	Optional	Transmit and receive every eighth send clock
16	Optional	Transmit and receive every 16th send clock
32	Optional	Transmit and receive every 32th send clock
64	Optional	Transmit and receive every 64th send clock
128	Mandatory	Transmit and receive every 128th send clock
256	Mandatory	Transmit and receive every 256th send clock
512	Mandatory	Transmit and receive every 512th send clock
1 024	Mandatory	Transmit and receive every 1 024th send clock
2 048	Mandatory	Transmit and receive every 2 048th send clock
4 096	Mandatory	Transmit and receive every 4 096th send clock

Value (decimal)	Meaning	Use
8 192	Mandatory	Transmit and receive every 8 192nd send clock
16 384	Mandatory	Transmit and receive every 16 384th send clock
16 385 – 65 535	Reserved	—
Other	Optional	Not intended to be used

5.2.4.60 Coding of the field Phase

This field shall be coded as data type Unsigned16 according to Table 646. It shall be less than or equal to the related ReductionRatio.

Table 646 – Values of Phase

Value (decimal)	Meaning	Use
0	Reserved	—
1 – 16 384	Mandatory	Shall not exceed the value of ReductionRatio
16 385 – 65 535	Reserved	—

5.2.4.61 Coding of the field Sequence

This field shall be coded as data type Unsigned16 according to Table 647.

Table 647 – Values of Sequence

Value (hexadecimal)	Meaning	Use
0	Mandatory, sequence undefined	The PPMs use a random sequence for each phase
0x0001 – 0xFFFF	Optional, sequence defined	The PPMs use the given sequence for each phase

NOTE For RT_CLASS_3 a given sequence is used, even if the parameter Sequence is set to zero.

5.2.4.62 Coding of the field DataHoldFactor

This field shall be coded as data type Unsigned16. The time base is SendClockFactor multiplied by the ReductionRatio of the monitored consumer.

The DataHoldTime shall be calculated according to Formula (62).

$$\text{DataHoldTime} = \text{DataHoldFactor} \times \text{SendClockFactor} \times \text{ReductionRatio} \times 31{,}25\ \mu\text{s} \quad (62)$$

where

DataHoldTime is the data hold time

DataHoldFactor is the data hold factor

SendClockFactor is the send clock factor

ReductionRatio is the reduction ratio

The value range for Data-RTC-PDUs is from 0x0003 to 0x1E00. The DataHoldTime shall be equal to or less than 1,92 s.

The value range for UDP-RTC-PDUs is from 0x0003 to 0xF000. The DataHoldTime shall be equal to or less than 61,44 s.

For a frame, the usage of the DataHoldFactor is defined in Table 648 and Table 649.

Table 648 – Data-RTC-PDUs – DataHoldFactor of a frame

Value (hexadecimal)	Meaning	Description
0x0000	Reserved	—
0x0001 – 0x0002	Optional	A value “1” leads to an AR termination for one missed frame, a value “2” leads to an AR termination for two missed frames.
0x0003 – 0x00FF	Mandatory	An expiration of the time leads to an AR termination.
0x0100 – 0x1E00	Optional	An expiration of the time leads to an AR termination.
0x1E01 – 0xFFFF	Reserved	—

Table 649 – UDP-RTC-PDUs – DataHoldFactor of a frame

Value (hexadecimal)	Meaning	Description
0x0000	Reserved	—
0x0001 – 0x0002	Optional	A value “1” leads to an AR termination for one missed frame, a value “2” leads to an AR termination for two missed frames.
0x0003 – 0x00FF	Mandatory	An expiration of the time leads to an AR termination.
0x0100 – 0xF000	Optional	An expiration of the time leads to an AR termination.
0xF001 – 0xFFFF	Reserved	—

For a Subframe, the usage of the DataHoldFactor is defined in Table 650.

Table 650 – DataHoldFactor of a Subframe

Value (hexadecimal)	Meaning	Description
0x0000 – 0x0002	Reserved	—
0x0003 – 0x001F	Mandatory	An expiration of the time leads to an AR termination.
0x0020 – 0xFFFF	Reserved	—

5.2.4.63 Coding of the field FrameSendOffset

This field shall be coded as data type Unsigned32 with values according to Table 651, Figure 172 and Formula (63). The time base shall be one nanosecond.

NOTE The minimal planned gap between two RT_CLASS_3 frames is 1 120 ns.

Table 651 – Values of FrameSendOffset

Value (hexadecimal)	Meaning	Use
0 – 0x00001388	5 000 ns Relative send offset from the start of the related cycle	Optional
0x00001389 – 0x003B28FF	3 877 119 ns Relative send offset from the start of the related cycle	Mandatory for RT_CLASS_3 Optional for RT_CLASS_x
0x003B2900 – 0x003D08FF	4 ms Relative send offset from the start of the related cycle Only usable if the frame size fulfills Formula (63)	Mandatory for RT_CLASS_3 Optional for RT_CLASS_x
0x003D0900 – 0xFFFFFFFE	Reserved	Not used
0xFFFFFFFF	Best effort Provider protocol machine sends the frame as soon as possible	Mandatory for RT_CLASS_x Not used for PDIRFRameData and RT_CLASS_3

$$\text{FrameSendOffset[Frame]} + \text{Duration[Frame]} < \text{SendClockFactor} \times 31{,}25\ \mu\text{s} \qquad (63)$$

where

FrameSendOffset is the frame send offset

Frame is the current frame

Duration is the duration

SendClockFactor is the factor of the send clock, the length of one phase

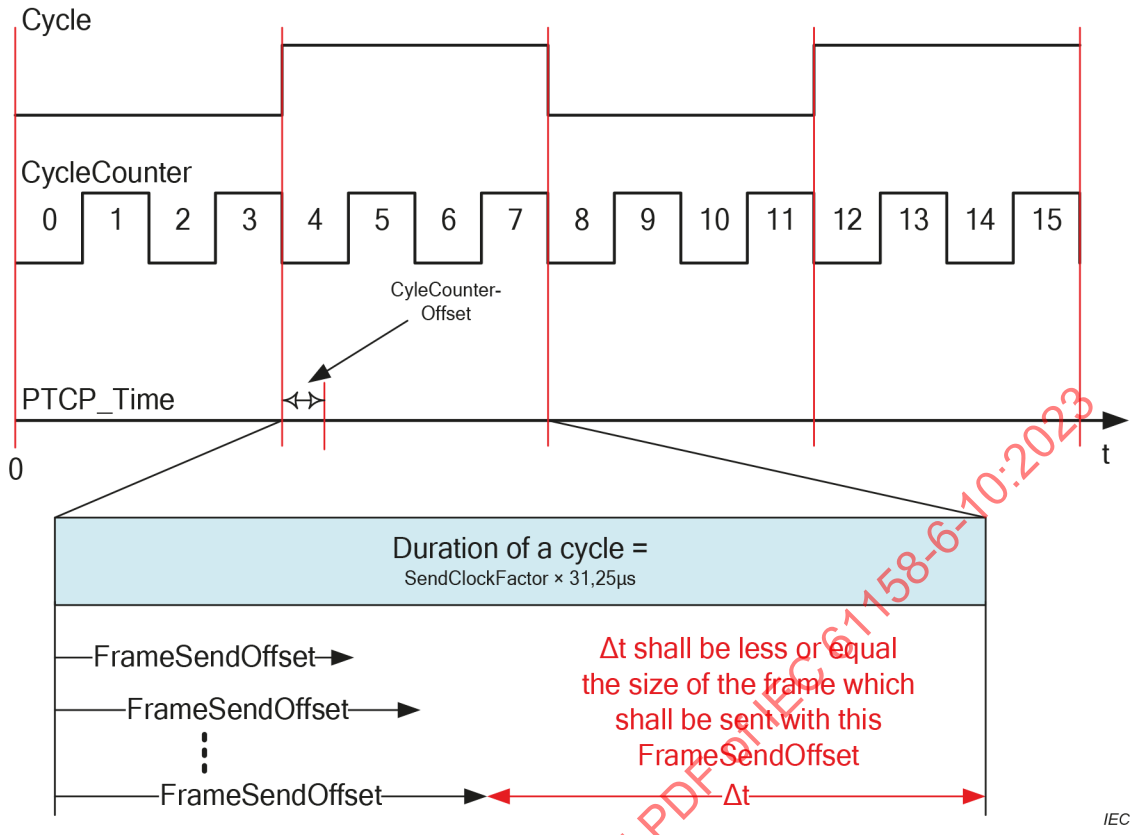


Figure 172 – FrameSendOffset vs. duration of a cycle

5.2.4.64 Coding of the field ModuleState

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 652.

Table 652 – ModuleState

Value (hexadecimal)	Meaning	Use
0x0000	NoModule	For example, module not plugged
0x0001	WrongModule [a]	For example, ModuleIdentNumber wrong
0x0002	ProperModule	Module is okay but at least one submodule is locked, wrong or missing
0x0003	Substitute [b]	Module is not the same as requested – but the IO device was able to adapt through its own knowledge
0x0004 – 0xFFFF	Reserved	—

[a] This coding should be avoided if possible. The substitution may be “Substitute” for the module and “Wrong (WR)” for each submodule requested by the ExpectedSubmoduleBlockReq. See Table 659 for valid combinations.

[b] This coding may be used in combination with “SubmoduleState.IdentInfo := Wrong (WR)”.

5.2.4.65 Coding of the field SubmoduleState

5.2.4.65.1 General

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning.

5.2.4.65.2 Coding if SubmoduleState.FormatIndicator == 1

Bit 0 – 2: SubmoduleState.AddInfo
This field shall be set according to Table 653.

Table 653 – SubmoduleState.AddInfo

Value (hexadecimal)	Meaning	Description
0x00	No additional info	None of the options defined for this field are activated.
0x01	Takeover is not allowed	This Submodule is not available for takeover by IOSAR.
0x02 – 0x07	Reserved	—

Bit 3: SubmoduleState.Advice
This field shall be set according to Table 654. It summarizes the Advice information of the Diagnosis ASE for the reporting submodule.

Table 654 – SubmoduleState.Advice

Value (hexadecimal)	Meaning	Description
0x00	No Advice information available	There is no Advice available/stored for this submodule.
0x01	Advice information available	At least one Advice is available/stored for this submodule.

Bit 4: SubmoduleState.MaintenanceRequired
This field shall be set according to Table 655. It summarizes the MaintenanceRequired information of the Diagnosis ASE for the reporting submodule.

Table 655 – SubmoduleState.MaintenanceRequired

Value (hexadecimal)	Meaning	Description
0x00	No MaintenanceRequired information available	There is no MaintenanceRequired available/stored for this submodule.
0x01	MaintenanceRequired information available	At least one requirement for maintenance is available/stored for this submodule.

Bit 5: SubmoduleState.MaintenanceDemanded
This field shall be set according to Table 656. It summarizes the MaintenanceDemanded information of the Diagnosis ASE for the reporting submodule.

Table 656 – SubmoduleState.MaintenanceDemanded

Value (hexadecimal)	Meaning	Description
0x00	No MaintenanceDemanded information available	There is no MaintenanceDemanded available/stored for this submodule.
0x01	MaintenanceDemanded information available	At least one demand for maintenance is available/stored for this submodule.

Bit 6: SubmoduleState.Fault
This field shall be set according to Table 657. It summarizes the Fault information of the Diagnosis ASE for the reporting submodule.

Table 657 – SubmoduleState.Fault

Value (hexadecimal)	Meaning	Description
0x00	No Fault information available	There is no Fault available/stored for this submodule.
0x01	Fault information available	At least one Fault is available/stored for this submodule.

Bit 7 – 10: SubmoduleState.ARInfo
This field shall be set according to Table 658.

Table 658 – SubmoduleState.ARInfo

Value (hexadecimal)	Meaning	Description
0x00	Own	This AR is owner of the submodule
0x01	ApplicationReadyPending (ARP)	This AR is owner of the submodule but it is blocked. For example parameter checking pending
0x02	Superordinated Locked (SO)	This AR is not owner of the submodule. It is blocked by superordinated means
0x03	Locked By IO Controller (IOC)	This AR is not owner of the submodule. It is owned by another IOAR
0x04	Locked By IO Supervisor (IOS)	This AR is not owner of the submodule. It is owned by another IOSAR
0x05 – 0x0F	Reserved	—

Bit 11 – 14: SubmoduleState.IdentInfo
This field shall be set according to Table 659.

Table 659 – SubmoduleState.IdentInfo

Value (hexadecimal)	Meaning	Usable in conjunction with ModuleState "Wrong module"
0x00	OK	No
0x01	Substitute (SU)	No
0x02	Wrong (WR)	Yes
0x03	NoSubmodule (NO)	Yes
0x04 – 0x0F	Reserved	—

Bit 15: SubmoduleState.FormatIndicator

This field shall be set according to Table 660.

Table 660 – SubmoduleState.FormatIndicator

Value (hexadecimal)	Meaning	Use
0x00	Coding uses SubmoduleState.Detail	Legacy May be supported by an IO controller and IO supervisor.
0x01	Coding uses SubmoduleState.IdentInfo, SubmoduleState.ARInfo and SubmoduleState.AddInfo	Shall be used by an IO device, IO controller and IO supervisor.

5.2.4.66 Coding of the field SubmoduleProperties

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: SubmoduleProperties.Type
This field shall be set according to Table 661.

Table 661 – SubmoduleProperties.Type

Value (hexadecimal)	Meaning	Use
0x00	No input and no output data NO_IO	Submodule without IO Data, treated as input without data, one Input DataDescription Block follows
0x01	Input data INPUT	Submodule with input data, one Input DataDescription Block follows
0x02	Output data OUTPUT	Submodule with output data, one Output DataDescription Block follows
0x03	Input and output data INPUT_OUTPUT	Submodule with input and output data, one Input and one Output DataDescription Block follow

Bit 2: SubmoduleProperties.SharedInput
This field shall be set according to Table 662.

Table 662 – SubmoduleProperties.SharedInput

Value (hexadecimal)	Meaning	Use
0x00	IO controller	IO controller Can be used together with all possible values of SubmoduleProperties.Type.
0x01	IO controller shared	Only the shared IO controller shall use this value. The shared IO controller does not receive alarms and is restricted to read access. The value shall not be used together with SubmoduleProperties.Type == OUTPUT.

Bit 3: SubmoduleProperties.ReduceInputSubmoduleDataLength
This field shall be set according to Table 663.

Table 663 – SubmoduleProperties.ReduceInputSubmoduleDataLength

Value (hexadecimal)	Meaning	Use
0x00	Expected	Use expected input SubmoduleDataLength for Input CR
0x01	Zero	Reduce input SubmoduleDataLength to zero for Input CR. Shall only be set if SubmoduleProperties.Type == INPUT or SubmoduleProperties.Type == INPUT_OUTPUT is used.

Bit 4: SubmoduleProperties.ReduceOutputSubmoduleDataLength
This field shall be set according to Table 664.

Table 664 – SubmoduleProperties.ReduceOutputSubmoduleDataLength

Value (hexadecimal)	Meaning	Use
0x00	Expected	Use expected output SubmoduleDataLength for Output CR
0x01	Zero	Reduce output SubmoduleDataLength to zero for Output CR. Shall only be set if SubmoduleProperties.Type == OUTPUT or SubmoduleProperties.Type == INPUT_OUTPUT is used.

Bit 5: SubmoduleProperties.DiscardIOXS
This field shall be set according to Table 665.

Table 665 – SubmoduleProperties.DiscardIOXS

Value (hexadecimal)	Meaning	Use
0x00	Expected	Merge all expected IOXS for this submodule in the frames of the corresponding CRs
0x01	Zero	Discard all expected IOXS for this submodule in the frames of the corresponding CRs. Shall only be set if SubmoduleProperties.Type == NO_IO is used.

Bit 6 – 7: SubmoduleProperties.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 8 – 15: SubmoduleProperties.Reserved_2
This field shall be set to zero.

5.2.4.67 Coding of the field ModuleProperties

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: ModuleProperties.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 8 – 15: ModuleProperties.Reserved_2
This field shall be set to zero.

5.2.4.68 Coding of the field SubstitutionMode

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 666.

Table 666 – SubstitutionMode

Value (hexadecimal)	Meaning	Use
0x0000	ZERO	The outputs are set to inactive, for example zero
0x0001	Last value	Hold the last valid application value
0x0002	Replacement value	Set the output to the configured replacement value
0x0003 – 0x00FF	Reserved	—
0x0100 – 0x01FF	Reserved for profiles	Shall be used for profiles
0x0200 – 0xFFFF	Reserved	—

5.2.4.69 Coding of the field SubstituteActiveFlag

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 667.

Table 667 – SubstituteActiveFlag

Value (hexadecimal)	Meaning	Use
0x0000	Operation	The outputs are set according to normal control process (substitute inactive)
0x0001	Substitute	The outputs are set according to substitute function (substitute active)
0x0002 – 0xFFFF	Reserved	—

5.2.4.70 Coding of the field InitiatorUDPRTPort

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 668.

Table 668 – InitiatorUDPRTPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Optional Port can be blocked by other protocols
0x0400 – 0x8891	IANA_DEFINED_PORT	Usable
0x8892	Default for unicast IANA_PNIO_UDP_UNICAST_PORT	Recommended
0x8893	Default for multicast IANA_PNIO_UDP_MULTICAST_PORT	Recommended
0x8894 – 0xBFFF	IANA_DEFINED_PORT	Usable
0xC000 – 0xFFFF	IANA_FREE_PORT is used as dynamic and/or private port	Usable

5.2.4.71 Coding of the field ResponderUDPRTPort

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 669.

Table 669 – ResponderUDPRTPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Optional Port can be blocked by other protocols
0x0400 – 0x8891	IANA_DEFINED_PORT	Usable
0x8892	Default for unicast IANA_PNIO_UDP_UNICAST_PORT	Recommended
0x8893	Default for multicast IANA_PNIO_UDP_MULTICAST_PORT	Recommended
0x8894 – 0xBFFF	IANA_DEFINED_PORT	Usable
0xC000 – 0xFFFF	IANA_FREE_PORT is used as dynamic and/or private port	Usable

5.2.4.72 Coding of the field InitiatorRPCServerPort

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 670.

Table 670 – InitiatorRPCServerPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Reserved
0x0400 – 0xBFFF	IANA_DEFINED_PORT	Usable
0xC000 – 0xFFFF	IANA_FREE_PORT is used as dynamic and/or private port	Recommended

5.2.4.73 Coding of the field ResponderRPCServerPort

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 671.

Table 671 – ResponderRPCServerPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Reserved
0x0400 – 0xBFFF	IANA_DEFINED_PORT	Usable
0xC000 – 0xFFFF	IANA_FREE_PORT is used as dynamic and/or private port	Recommended

5.2.4.74 Coding of the field ResponderActivityUUID

This field shall be coded according to 4.10.3.2.10. This field contains the value of the RPCActivityUUID used by the responder for the service application ready in the context of this AR.

5.2.4.75 Coding of the field LocalAlarmReference

This field shall be coded as data type Unsigned16. This field shall contain the value of the reference of the IO controller within AlarmCRBlockReq and shall contain the value of the reference of the IO device within AlarmCRBlockRes.

In the ARData, this field contains the reference of the IO controller received by the AlarmCRBlockReq.

5.2.4.76 Coding of the field RemoteAlarmReference

This field shall be coded as data type Unsigned16. This field shall contain the value of the reference of the IO device, transmitted within AlarmCRBlockRes to the IO controller, within ARData.

5.2.4.77 Coding of the field MaxAlarmDataLength

This field shall be coded as data type Unsigned16 with values according to Table 672.

Table 672 – MaxAlarmDataLength

Value (hexadecimal)	Meaning
0x00C8	Mandatory Maximum size of the AlarmNotification-PDU accepted by the AlarmEndpoint.
0x00C9 – 0x0598	Optional Maximum size of the AlarmNotification-PDU accepted by the AlarmEndpoint.
Other	Reserved

An AlarmEndpoint shall not create larger AlarmNotification-PDUs than negotiated during AR establishment.

5.2.4.78 Coding of the field ParameterServerProperties

This field shall be coded as data type Unsigned32. This field is reserved for future use.

5.2.5 Coding section related to ARVendorBlock

5.2.5.1 Coding of the field APStructureIdentifier

This field, defined as an administrative number, shall be coded as data type Unsigned16 according to Table 673 and Table 674.

Table 673 – APStructureIdentifier with API==0

Value (hexadecimal)	Meaning
0 – 0x7FFF	Vendor specific This key defines in combination with the key API the structure of Data in ARVendorBlockReq and ARVendorBlockRes.
0x8000	Extended identification rules For example used for common profile “Controller to controller communication”
0x8001 – 0xFFFF	Administrative number This key defines in combination with the key API the structure of Data in ARVendorBlockReq and ARVendorBlockRes.

Table 674 – APStructureIdentifier with API ≠ 0

Value (hexadecimal)	Meaning
0 – 0xFFFF	Administrative number for application profiles This key defines in combination with the key API the structure of Data in ARVendorBlockReq and ARVendorBlockRes.

5.2.5.2 Coding of the field ExtendedIdentificationUUID

This field shall be coded as data type UUID.

5.2.5.3 Coding of the field ExtendedIdentificationVersionHigh

This field shall be coded as data type Unsigned8 with the values according to Table 675.

Table 675 – ExtendedIdentificationVersionHigh

Value (hexadecimal)	Meaning	Use
0x00	Reserved	—
0x01	Version 1	Indicates version 1
0x02 – 0xFF	Version 2 to version 255	Indicates version 2 to 255

5.2.5.4 Coding of the field ExtendedIdentificationVersionLow

This field shall be coded as data type Unsigned8 with the values according to Table 676.

Table 676 – ExtendedIdentificationVersionLow

Value (hexadecimal)	Meaning	Use
0x00	Version 0	Indicates version 0
0x01	Version 1	Indicates version 1
0x02 – 0xFF	Version 2 to version 255	Indicates version 2 to 255

5.2.6 Coding section related to PNIOStatus

5.2.6.1 General

In general, the values ErrorCode=0, ErrorDecode=0, ErrorCode1=0 and ErrorCode2=0 shall be used to indicate “okay”.

Furthermore, in case of an illegal combination of address parameters within an IODReadReq the values ErrorCode=“IODReadRes”, ErrorDecode=“PNIORW”, ErrorCode1=“access-invalid area” and ErrorCode2 may be used to indicate the faulty parameter.

NOTE An illegal address combination is for example TargetARUUID == NIL and ARUUID == NIL in ReadExpectedIdentification service.

5.2.6.2 Coding of the field ErrorCode

This field shall be coded as data type Unsigned8. The usage is defined in Table 677.

Table 677 – Values of ErrorCode for negative responses

Value (hexadecimal)	Meaning	Use
0x00	Reserved	Special case “No Error”: ErrorCode = 0, ErrorDecode = 0, ErrorCode1 = 0, ErrorCode2 = 0
0x01 – 0x1F	Reserved	—
0x20 – 0x3F	Manufacturer specific	Within the LogBookData
0x40 – 0x80	Reserved	—
0x81	PNIO	Used for all errors not covered elsewhere.
0x82 – 0xCE	Reserved	—
0xCF	RTA error	Within the ERR-RTA-PDU and UDP-RTA-PDU
0xD0 – 0xD9	Reserved	—
0xDA	AlarmAck	Within the DATA-RTA-PDU and UDP-RTA-PDU
0xDB	IODConnectRes	Within the CL-RPC-PDU
0xDC	IODReleaseRes	Within the CL-RPC-PDU
0xDD	IODControlRes, IOXControlRes	Within the CL-RPC-PDU
0xDE	IODReadRes	Within the CL-RPC-PDU
0xDF	IODWriteRes	Within the CL-RPC-PDU
0xE0 – 0xEF	Reserved	—
0xF0 – 0xFF	Reserved	—

5.2.6.3 Coding of the field ErrorDecode

This field shall be coded as data type Unsigned8. The usage is defined in Table 678.

Table 678 – Values of ErrorDecode

Value (hexadecimal)	Meaning	Use
0x00	Reserved	See Table 677
0x01 – 0x7F	Reserved	—
0x80	PNIORW [a]	Used in context with user error codes of the services Read and Write
0x81	PNIO	Used in context with other services or internal, for example RPC errors
0x82	Manufacturer specific	Used only in context of LogBookData
0x83 – 0xFF	Reserved	—
[a] Equivalent to IEC 61158-6-3 (DPV1)		

5.2.6.4 Coding of the field ErrorCode1 and ErrorCode2

5.2.6.4.1 General

These fields shall be coded as data type Unsigned8.

The field ErrorDecode defines the coding and meaning of ErrorCode1 and ErrorCode2.

5.2.6.4.2 ErrorDecode value PNIORW

The ErrorDecode value PNIORW indicates that the parameters ErrorCode1 shall be set according to Table 679. The ErrorCode1 consists of ErrorClass and ErrorDecode. See also error code hierarchy in IEC 61158-5-10.

Table 679 – Coding of ErrorCode1 with ErrorDecode PNIORW

ErrorCode1		
ErrorClass (decimal) Bit7 – 4	Meaning	ErrorCode (decimal) Bit3 – 0
0 to 9	Not specified	Reserved
10	Application	0 = read error 1 = write error 2 = module failure 3 = not specified 4 = not specified 5 = not specified 6 = not specified 7 = busy 8 = version conflict 9 = feature not supported 10 = User specific 1 11 = User specific 2 12 = User specific 3 13 = User specific 4 14 = User specific 5 15 = User specific 6

ErrorCode1		
ErrorClass (decimal) **Bit7 – 4**	**Meaning**	**ErrorCode (decimal)** **Bit3 – 0**
11	Access	0 = invalid index 1 = write length error 2 = invalid slot / subslot 3 = type conflict 4 = invalid area / API 5 = state conflict 6 = access denied 7 = invalid range 8 = invalid parameter 9 = invalid type 10= backup 11 = User specific 7 12 = User specific 8 13 = User specific 9 14 = User specific 10 15 = User specific 11
12	Resource	0 = read constraint conflict 1 = write constraint conflict 2 = resource busy 3 = resource unavailable 4 = not specified 5 = not specified 6 = not specified 7 = not specified 8 = User specific 12 9 = User specific 13 10 = User specific 14 11 = User specific 15 12 = User specific 16 13 = User specific 17 14 = User specific 18 15 = User specific 19
13 to 15	User specific	User specific
NOTE Not specified values are used to serve legacy codes and are intended to be passed unchanged to the application.		

The ErrorDecode value PNIORW indicates that the parameter ErrorCode2 shall be set according to Table 680.

Table 680 – Coding of ErrorCode2 with ErrorDecode PNIORW

ErrorCode2 (hexadecimal)	Meaning	Usage
0x00 – 0xFF	User specific	—

5.2.6.4.3 ErrorDecode value PNIO

5.2.6.4.3.1 ErrorCode1

Table 681 shows the values for ErrorCode1 together with ErrorDecode := PNIO.

Table 681 – Coding of ErrorCode1 with ErrorDecode := PNIO

ErrorCode1 (hexadecimal)	Meaning
0x00	Reserved
0x01	Connect Parameter Error Faulty ARBlockReq
0x02	Connect Parameter Error Faulty IOCRBlockReq
0x03	Connect Parameter Error Faulty ExpectedSubmoduleBlockReq
0x04	Connect Parameter Error Faulty AlarmCRBlockReq
0x05	Connect Parameter Error Faulty PrmServerBlock
0x06	Connect Parameter Error Faulty MCRBlockReq
0x07	Connect Parameter Error Faulty ARRPCBlockReq
0x08	Read Write Record Parameter Error Faulty Record
0x09	Connect Parameter Error Faulty IRInfoBlock
0x0A	Connect Parameter Error Faulty SRInfoBlock
0x0B	Connect Parameter Error Faulty ARFSUBlock
0x0C	Connect Parameter Error Faulty ARVendorBlockReq
0x0D	Connect Parameter Error Faulty RSInfoBlock
0x0E	Connect Parameter Error Faulty ARAlgorithmInfoBlock
0x0F – 0x13	Reserved
0x14	IODControl Parameter Error Faulty ControlBlockConnect
0x15	IODControl Parameter Error Faulty ControlBlockPlug
0x16	IOXControl Parameter Error Faulty ControlBlockConnect after a connection establishment
0x17	IOXControl Parameter Error Faulty ControlBlockPlug after a plug alarm
0x18	IODControl Parameter Error Faulty ControlBlockPrmBegin
0x19	IODControl Parameter Error Faulty SubmoduleListBlock
0x1A – 0x27	Reserved
0x28	Release Parameter Error Faulty ReleaseBlock
0x29 – 0x31	Reserved

ErrorCode1 (hexadecimal)	Meaning
0x32	Response Parameter Error Faulty ARBlockRes
0x33	Response Parameter Error Faulty IOCRBlockRes
0x34	Response Parameter Error Faulty AlarmCRBlockRes
0x35	Response Parameter Error Faulty ModuleDiffBlock
0x36	Response Parameter Error Faulty ARRPCBlockRes
0x37	Response Parameter Error Faulty ARServerBlockRes
0x38	Response Parameter Error Faulty ARVendorBlockRes
0x39 – 0x3B	Reserved
0x3C	AlarmAck Error Codes
0x3D	CMDEV
0x3E	CMCTL
0x3F	CTLDINA
0x40	CMRPC
0x41	ALPMI
0x42	ALPMR
0x43	LMPM
0x44	MAC
0x45	RPC
0x46	APMR
0x47	APMS
0x48	CPM
0x49	PPM
0x4A	DCPUCS
0x4B	DCPUCR
0x4C	DCPMCS
0x4D	DCPMCR
0x4E	FSPM
0x4F	RSII
0x50	RSIR
0x51 – 0x63	Reserved
0x64	CTLSM
0x65	CTLRDI
0x66	CTLRDR
0x67	CTLWRI
0x68	CTLWRR
0x69	CTLIO

ErrorCode1 (hexadecimal)	Meaning
0x6A	CTLSU
0x6B	CTLRPC
0x6C	CTLPBE
0x6D	CTLSRL
0x6E	NME
0x6F	TDE
0x70	PCE
0x71	NCE
0x72	NUE
0x73	BNME
0x74	CTLSAM
0x75 – 0xC7	Reserved
0xC8	CMSM
0xC9	Reserved
0xCA	CMRDR
0xCB	Reserved
0xCC	CMWRR
0xCD	CMIO
0xCE	CMSU
0xCF	Reserved
0xD0	CMINA
0xD1	CMPBE
0xD2	CMSRL
0xD3	CMDMC
0xD4	CMSAM
0xD5 – 0xFC	Reserved
0xFD	Used by RTA for protocol error (RTA_ERR_CLS_PROTOCOL)
0xFE	Reserved
0xFF	User specific

5.2.6.4.3.2 ErrorCode2 values for dedicated ErrorCode1 values

The ErrorDecode value PNIO indicates that the parameter ErrorCode1 shall be set according to Table 682, Table 683, Table 684, Table 685, Table 686, Table 687, Table 688 and the definitions of 4.10.3.4.6. For usage, see 5.8 containing the PDU checking rules.

Table 682 – Values of ErrorCode2 for ErrorDecode := PNIO and ErrorCode1 (part 1)

Meaning	Value ErrorCode2 (hexadecimal)	Meaning/Usage
Connect Parameter Error Faulty ARBlockReq	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x0D	Error in Parameter NameOfStation
	0x0E – 0xFF	Reserved
Connect Parameter Error Faulty IOCRBlockReq	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x1C	Error in Parameter IOCSFrameOffset production
	0x1D – 0xFF	Reserved
Connect Parameter Error Faulty ExpectedSubmoduleBlockReq	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x10	Error in Parameter LengthIOCS production
	0x11 – 0xFF	Reserved
Connect Parameter Error Faulty AlarmCRBlockReq	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x0E	Error in Parameter AlarmCRTagHeaderLow
	0x10 – 0xFF	Reserved
Connect Parameter Error Faulty PrmServerBlock	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x08	Error in Parameter ParameterServerStationName
	0x09 – 0xFF	Reserved
Connect Parameter Error Faulty MCRBlockReq	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x08	Error in Parameter ProviderStationName
	0x09 – 0xFF	Reserved
Connect Parameter Error Faulty ARRPCBlockReq	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x04	Error in Parameter InitiatorRPCServerPort
	0x05 – 0xFF	Reserved

Meaning	Value ErrorCode2 (hexadecimal)	Meaning/Usage
Read Write Record Parameter Error Faulty Record	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x0C	Error in Parameter TargetARUUID
	0x0D – 0xFF	Reserved
Connect Parameter Error Faulty IRInfoBlock	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x05	Error in Parameter IRDataUUID
	0x06 – 0xFF	Reserved
Connect Parameter Error Faulty SRInfoBlock	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x05	Error in Parameter SRProperties
	0x06 – 0xFF	Reserved
Connect Parameter Error Faulty ARFSUBlock	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x05	FastStartUpBlock
	0x06 – 0xFF	Reserved
Connect Parameter Error Faulty ARVendorBlockReq	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x05	Error in Parameter API
	0x06	Error in Parameter Data*
	0x07 – 0xFF	Reserved
Connect Parameter Error Faulty RSInfoBlock	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x05	Error in Parameter RSProperties
	0x06 – 0xFF	Reserved
Connect Parameter Error Faulty ARAlgorithmInfoBlock	0x00	Error in Parameter BlockType
	…	…
	0x08 – 0x0C	Reserved for security
	0x0D – 0xFF	Reserved
IODControl Parameter Error Faulty ControlBlockConnect	0x01	Error in Parameter Length
	…	…
	0x05	Error in Parameter RSProperties
	0x06 – 0xFF	Reserved
	0x0A – 0xFF	Reserved

Meaning	Value ErrorCode2 (hexadecimal)	Meaning/Usage
IODControl Parameter Error Faulty ControlBlockPlug	0x00	Error in Parameter BlockType
	…	…
	0x09	Error in Parameter ControlBlockProperties
	0x0A – 0xFF	Reserved
IOXControl Parameter Error Faulty ControlBlockConnect after a connection establishment	0x00	Error in Parameter BlockType
	…	…
	0x07	Error in Parameter ControlBlockProperties
	0x08 – 0xFF	Reserved
IOXControl Parameter Error Faulty ControlBlockPlug after a plug alarm	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x07	Error in Parameter ControlBlockProperties
	0x08 – 0xFF	Reserved
IODControl Parameter Error Faulty ControlBlockPrmBegin	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x09	Error in Parameter ControlBlockProperties
	0x0A – 0xFF	Reserved
IODControl Parameter Error Faulty SubmoduleListBlock	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x07	Error in Parameter SubslotNumber
	0x08 – 0xFF	Reserved
Release Parameter Error Faulty ReleaseBlock	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x07	Error in Parameter ControlBlockProperties
	0x08 – 0xFF	Reserved
Response Parameter Error Faulty ARBlockRes	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x08	Error in Parameter ResponderUDPRTPort
	0x09 – 0xFF	Reserved
Response Parameter Error Faulty IOCRBlockRes	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x06	Error in Parameter FrameID
	0x07 – 0xFF	Reserved

Meaning	Value ErrorCode2 (hexadecimal)	Meaning/Usage
Response Parameter Error Faulty AlarmCRBlockRes	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x06	Error in Parameter MaxAlarmDataLength
	0x07 – 0xFF	Reserved
Response Parameter Error Faulty ModuleDiffBlock	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x0D	Error in Parameter SubmoduleState
	0x0E – 0xFF	Reserved
Response Parameter Error Faulty ARRPCBlockRes	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x04	Error in Parameter ResponderRPCServerPort
	0x05 – 0xFF	Reserved
Response Parameter Error Faulty ARServerBlockRes	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x04	Error in Parameter CMResponderStationName
	0x05 – 0xFF	Reserved
Response Parameter Error Faulty ARVendorBlockRes	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	…	…
	0x05	Error in Parameter API
	0x06	Error in Parameter Data*
	0x07 – 0xFF	Reserved

Table 683 – Values of ErrorCode2 for ErrorDecode := PNIO and ErrorCode1 (part 2 – alarm acknowledge)

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
AlarmAck Error Codes	0x00	Alarm Type Not Supported
	0x01	Wrong Submodule State
	0x02	IOCARSR Backup – Alarm not executed
	0x03 – 0xFF	Reserved

Table 684 – Values of ErrorCode2 for ErrorDecode := PNIO and ErrorCode1 (part 3 – machines)

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
ALPMI	0x00	Invalid state
	0x01	Wrong ACK-PDU
	0x02	Invalid
	0x03	Wrong state
	0x04 – 0xFF	Reserved
ALPMR	0x00	Invalid state
	0x01	Wrong Notification PDU
	0x02	Invalid
	0x03	Wrong state
	0x04 – 0xFF	Reserved
LMPM	0x00 – 0xFF	Usage see protocol machines and stored as LogBook entries
MAC	0x00 – 0xFF	Usage see protocol machines and stored as LogBook entries
RPC	0x00	Reserved
	0x01	Endpoint mapper or server did reject the call. For further details see Table 346 and Table 347. (CLRPC_ERR_REJECTED)
	0x02	Server had a fault while executing the call. For further details see Table 346 and Table 347. (CLRPC_ERR_FAULTED)
	0x03	Endpoint mapper or server did not respond (CLRPC_ERR_TIMEOUT)
	0x04	Broadcast or maybe “ndr_data” too large (CLRPC_ERR_IN_ARGS)
	0x05	Server sent back more than “alloc_len” (CLRPC_ERR_OUT_ARGS)
	0x06	Result of endpoint mapper “lookup” could not be decoded (CLRPC_ERR_DECODE)
	0x07	(CLRPC_ERR_PNIO_OUT_ARGS)
	0x08	RPC call was terminated after RPC application timeout (CLRPC_ERR_PNIO_APP_TIMEOUT)
	0x09 to 0xFF	Reserved
APMR	0x00	Invalid state
	0x01	LMPM signaled an error
	0x02 – 0xFF	Reserved

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
APMS	0x00	Invalid state
	0x01	LMPM signaled an error
	0x02	Timeout
	0x03 – 0xFF	Reserved
CPM	0x00	Invalid state
	0x01 – 0xFF	Reserved
PPM	0x00	Invalid state
	0x01 – 0xFF	Reserved
DCPUCS	0x00	Invalid state
	0x01	LMPM signaled an error
	0x02	Timeout
	0x03 – 0xFF	Reserved
DCPUCR	0x00	Invalid state
	0x01	LMPM signaled an error
	0x02 – 0xFF	Reserved
DCPMCS	0x00	Invalid state
	0x01	LMPM signaled an error
	0x02 – 0xFF	Reserved
DCPMCR	0x00	Invalid state
	0x01	LMPM signaled an error
	0x02 – 0xFF	Reserved
FSPM	0x00 – 0xFF	Usage see protocol machines and stored as LogBook entries
RSII	0x00	State conflict
	0x01	Abort
	0x02 – 0xFF	Reserved
RSIR	0x00	State conflict
	0x01	Abort
	0x02	InterfaceNotFound
	0x03	OutOfResources
	0x04	Rerun connect
	0x05	RspMaxLength
	0x06 – 0xFF	Reserved

Table 685 – Values of ErrorCode2 for ErrorDecode := PNIO and ErrorCode1 (part 4 – IO controller)

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
CMCTL	0x00	State conflict
	0x01	Timeout
	0x02	No data send
	0x03	Out of resource
	0x04 – 0xFF	Reserved
CTLDINA	0x00	No DCP active / No Link
	0x01	DNS unknown RealStationName
	0x02	DCP no RealStationName
	0x03	DCP multiple RealStationName
	0x04	DCP no StationName
	0x05	No IP address
	0x06	DCP set error
	0x07	ARP multiple IP-Addresses
	0x08 – 0xFF	Reserved
CTLSM	0x00	Invalid state
	0x01	CTLSM signaled an error
	0x02 – 0xFF	Reserved
CTLRDI	0x00	Invalid state
	0x01	CTLRDI signaled an error
	0x02 – 0xFF	Reserved
CTLRDR	0x00	Invalid state
	0x01	CTLRDR signaled an error
	0x02 – 0xFF	Reserved
CTLWRI	0x00	Invalid state
	0x01	CTLWRI signaled an error
	0x02 – 0xFF	Reserved
CTLWRR	0x00	Invalid state
	0x01	CTLWRR signaled an error
	0x02 – 0xFF	Reserved
CTLIO	0x00	Invalid state
	0x01	CTLIO signaled an error
	0x02 – 0xFF	Reserved
CTLSU	0x00	Invalid state
	0x01	AR add provider or consumer failed
	0x02	AR alarm-open failed
	0x03	AR alarm-ack-send
	0x04	AR alarm-send
	0x05	AR alarm-ind
	0x06 – 0xFF	Reserved

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
CTLRPC	0x00	Invalid state
	0x01	CTLRPC signaled an error
	0x02 – 0xFF	Reserved
CTLPBE	0x00	Invalid state
	0x01	CTLPBE signaled an error
	0x02 – 0xFF	Reserved
CTLSRL	0x00	Invalid state
	0x01	CTLSRL signaled an error
	0x02 – 0xFF	Reserved
NME	0x00	Invalid state
	0x01	Temporary unknown
	0x02	Not best NME
	0x03 – 0xFF	Reserved
TDE	0x00	Invalid state
	0x01 – 0xFF	Reserved
PCE	0x00	Invalid state
	0x01	No path found
	0x02 – 0xFF	Reserved
NCE	0x00	Invalid state
	0x01 – 0xFF	Reserved
NUE	0x00	Invalid state
	0x01	Remote problem
	0x02 – 0xFF	Reserved
BNME	0x00	Invalid state
	0x01 – 0xFF	Reserved
CTLSAM	0x00	Invalid state
	0x01 – 0xFF	Reserved

Table 686 – Values of ErrorCode2 for ErrorDecode := PNIO and ErrorCode1 (part 5 – IO device)

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
CMDEV	0x00	State conflict
	0x01	Resource
	0x02 – 0xFF	Usage see protocol machines and stored as LogBook entries
CMRPC	0x00	ArgsLength invalid
	0x01	Unknown blocks
	0x02	IOCR missing
	0x03	Wrong block count (for example wrong AlarmCRBlock count)
	0x04	Out of AR resources
	0x05	AR UUID unknown
	0x06	State conflict
	0x07	Out of Provider, Consumer, or Alarm Resources
	0x08	Out of memory
	0x09	Pdev already owned
	0x0A	ARset State conflict during connection establishment
	0x0B	ARset Parameter conflict during connection establishment
	0x0C	Pdev, port(s) without interface
	0x0D	Discrepancy between Pdev and ARType
	0x0E	Pdev Discrepancy between Pdev ownership and ARProperties
	0x0F – 0xFF	Reserved
CMSM	0x00	Invalid state
	0x01	CMSM signaled an error
	0x02 – 0xFF	Reserved
CMRDR	0x00	Invalid state
	0x01	CMRDR signaled an error
	0x02 – 0xFF	Reserved
CMWRR	0x00	Invalid state
	0x01	AR is not in state Primary. In this case the write of a record is not allowed.
	0x02	CMWRR signaled an error
	0x03 – 0xFF	Reserved
CMIO	0x00	Invalid state
	0x01	CMIO signaled an error
	0x02 – 0xFF	Reserved

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
CMSU	0x00	Invalid state
	0x01	AR add provider or consumer failed
	0x02	AR alarm-open failed
	0x03	AR alarm-send
	0x04	AR alarm-ack-send
	0x05	AR alarm-ind
	0x06 – 0xFF	Reserved
CMINA	0x00	Invalid state
	0x01	CMINA signaled an error
	0x02 – 0xFF	Reserved
CMPBE	0x00	Invalid state
	0x01	CMPBE signaled an error
	0x02 – 0xFF	Reserved
CMSRL	0x00	Invalid state
	0x01	CMSRL signaled an error
	0x02 – 0xFF	Reserved
CMDMC	0x00	Invalid state
	0x01	CMDMC signaled an error
	0x02 – 0xFF	Reserved
CMSAM	0x00	Invalid state
	0x01 – 0xFF	Reserved

Table 687 – Values of ErrorCode2 for ErrorDecode := PNIO and ErrorCode1 (part 6 – abort reasons)

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
Used by RTA for protocol error (RTA_ERR_CLS_PROTOCOL)	0x00	Reserved
	RTA_ERR_CODE_SEQ	
	0x01	Error within the coordination of sequence numbers
	RTA_ERR_ABORT	
	0x02	Instance closed
	0x03	AR out of memory
	0x04	AR add provider or consumer failed
	0x05	AR consumer DHT expired
	0x06	AR CMI timeout
	0x07	AR alarm-open failed
	0x08	AR alarm-send.cnf (-)
	0x09	AR alarm-ack-send.cnf (-)
	0x0A	AR alarm data too long
	0x0B	AR alarm.ind (err)
	0x0C	AR rpc-client call.cnf (-)

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
	0x0D	AR abort.req
	0x0E	AR re-run aborts existing
	0x0F	AR release.ind received
	0x10	AR device deactivated
	0x11	AR removed
	0x12	AR protocol violation
	0x13	AR name resolution error
	0x14	AR RPC-Bind error
	0x15	AR RPC-Connect error
	0x16	AR RPC-Read error
	0x17	AR RPC-Write error
	0x18	AR RPC-Control error
	0x19	AR forbidden pull or plug after check.rsp and before in-data.ind
	0x1A	AR AP removed
	0x1B	AR link down
	0x1C	AR could not register multicast MAC address
	0x1D	Not synchronized (cannot start companion-AR)
	0x1E	Wrong topology (cannot start companion-AR)
	0x1F	DCP station name changed
	0x20	DCP reset to factory or factory reset
	0x21	Cannot start companion-AR because a 0xLIPP submodule in the first AR...
	0x22	No IRDATA record yet
	0x23	PDEV owner is leaving
	0x24	PDEV no port offers required speed/duplex mode
	0x25	IP-Suite [of the IOC] changed by means of DCP_set (IPParameter) or local engineering
	0x26	IOCARSR RDHT expired
	0x27	IOCARSR PDEV, parameterization impossible
	0x28	Remote application ready timeout expired
	0x29	IOCARSR Redundant interface lost or access to the peripherals impossible
	0x2A	IOCARSR MTOT expired

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
	0x2B	IOCARSR AR protocol violation
	0x2C	PDEV Plug port without CombinedObjectContainer
	0x2D	NME no or wrong configuration
	Reserved	
	0x2E – 0xC8	Reserved
	0xC9 – 0xFF	Manufacturer specific
User specific	0x00 – 0xFE	User specific
	0xFF	Recommended for "User abort" without further detail

Table 688 – Values of ErrorCode2 for ErrorDecode := PNIO and ErrorCode1 (part 7 – Reserved)

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
Reserved	0x00 – 0xFF	Reserved

5.2.6.4.4 ErrorDecode value ManufacturerSpecific

The ErrorDecode value ManufacturerSpecific indicates that the parameter ErrorCode1 and ErrorCode2 shall be set according to Table 689 and Table 690.

Table 689 – Coding of ErrorCode1 for ErrorDecode with the value ManufacturerSpecific

ErrorCode1 (hexadecimal)	Meaning	Usage
0x00 – 0xFF	Manufacturer specific	—

Table 690 – Coding of ErrorCode2 for ErrorDecode with the value ManufacturerSpecific

ErrorCode2 (hexadecimal)	Meaning	Usage
0x00 – 0xFF	Manufacturer specific	—

5.2.7 Coding section related to I&M Records

5.2.7.1 General

All data with type VisibleString shall be left justified. If the text is shorter than the defined string length, the gap shall be filled with blanks.

The engineering shall only use the visible characters for type VisibleString according to Table 691.

Table 691 – Visible characters

VisibleString (hexadecimal)	Usage
0x00 – 0x1F	Control characters
0x20 – 0x7E	Visible characters
0x7F	Control character

5.2.7.2 Reset behavior

Either Table 692 or Table 693 show the behavior after FactoryReset and ResetToFactory depending on the source of the named fields. Table 694 covers the special case usage of I&M4 together with functional safety submodules.

Table 692 – FactoryReset / ResetToFactory behavior (legacy from IEC 61158-6-3)

Field	Behavior
IM_Tag_Function	VisibleString[32] filled with the character blank ' '
IM_Tag_Location	VisibleString[22] filled with the character blank ' '
IM_Date	VisibleString[16] filled with the character blank ' '
IM_Descriptor	VisibleString[54] filled with the character blank ' '
IM_Signature	OctetString[54] filled with zero

Table 693 – FactoryReset / ResetToFactory behavior (default without IEC 61158-6-3 history)

Field	Behavior
IM_Tag_Function	Object shall be set to empty / unused
IM_Tag_Location	Object shall be set to empty / unused
IM_Date	Object shall be set to empty / unused
IM_Descriptor	Object shall be set to empty / unused
IM_Signature	Object shall be set to empty / unused

Table 694 – FactoryReset / ResetToFactory behavior if used in conjuction with functional safety submodules

Field	Behavior
IM_Signature	Defined by functional safety

5.2.7.3 Coding of the field IM_Serial_Number

This field shall be coded as data type VisibleString[16]. The value shall be set by the manufacturer according to 5.2.7.1.

This field uniquely identifies one produced entity and is thus an extension of

– VendorID, DeviceID, ModuleIdentNumber, and SubmoduleIdentNumber and/or
– VendorID, DeviceID, OrderID, HWRevision

5.2.7.4 Coding of the field IM_Hardware_Revision

This field shall be coded as data type Unsigned16 with the values according to Table 695.

Table 695 – IM_Hardware_Revision

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Hardware revision

5.2.7.5 Coding of the field IM_SWRevision_Functional_Enhancement

This field shall be coded as data type Unsigned8 with the values according to Table 696.

Table 696 – IM_SWRevision_Functional_Enhancement

Value (hexadecimal)	Meaning
0x00 – 0xFF	Functional Enhancement

5.2.7.6 Coding of the field IM_SWRevision_Bug_Fix

This field shall be coded as data type Unsigned8 with the values according to Table 697.

Table 697 – IM_SWRevision_Bug_Fix

Value (hexadecimal)	Meaning
0x00 – 0xFF	Bug Fix

5.2.7.7 Coding of the field IM_SWRevision_Internal_Change

This field shall be coded as data type Unsigned8 with the values according to Table 698.

Table 698 – IM_SWRevision_Internal_Change

Value (hexadecimal)	Meaning
0x00 – 0xFF	Internal Change

5.2.7.8 Coding of the field IM_Revision_Counter

This field shall be coded as data type Unsigned16 with the values according to Table 699.

Table 699 – IM_Revision_Counter

Value (hexadecimal)	Meaning	Usage
0x0000	Revision Counter	Mandatory
0x0001 – 0xFFFF	Revision Counter	Optional May be defined by application profiles, for example Process automation.

NOTE The IM_Revision_Counter has no meaning for the SWRevision.

5.2.7.9 Coding of the field IM_Profile_ID

This field, defined as an administrative number, shall be coded as data type Unsigned16 with the values according to Table 700. This field and the field API (see 5.2.3.1) are related.

Table 700 – IM_Profile_ID

Value (hexadecimal)	Meaning
0x0000	Non-profile device
0x0001 – 0xF6FF	Administrative number
0xF700 – 0xFFFE	Reserved for profiles
0xFFFF	Reserved

5.2.7.10 Coding of the field IM_Profile_Specific_Type

This field, defined as an administrative number, shall be coded as data type Unsigned16 with the values according to Table 701 and Table 702.

Table 701 – IM_Profile_Specific_Type in conjunction with IM_Profile_ID == 0x0000

Value (hexadecimal)	Meaning
0x0000	Unspecified
0x0001	Standard Controller, for example Programmable Logic Controller
0x0002	PC-based Station
0x0003	IO-Module or IO-Submodule
0x0004	Communication Module or Communication Submodule
0x0005	Interface Module or Interface Submodule
0x0006	Active Network Infrastructure Component
0x0007	Media Attachment Unit, for example an adapter to connect different Ethernet MAU types
0x0100 – 0x7FFF	Shall be defined by the manufacturer of the reporting entity
Other	Reserved

Table 702 – IM_Profile_Specific_Type in conjunction with IM_Profile_ID range 0x0001 – 0xF6FF

Value (hexadecimal)	Meaning
0x0000	Unspecified, if not defined otherwise by the profile
0x0001 – 0x00FF	Reserved [a]
0x0100 – 0x7FFF	Shall be defined by the manufacturer of the reporting entity
0x8000 – 0xFFFF	Using shall be defined by the profile identified by the IM_Profile_ID
[a] Profiles defined for IEC 61158 Type 3 may as well use this range due to legacy definitions	

5.2.7.11 Coding of the field IM_Version_Major

This field shall be coded as data type Unsigned8 with the values according to Table 703.

Table 703 – IM_Version_Major

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Shall be set in this version
0x02 – 0xFF	Reserved

5.2.7.12 Coding of the field IM_Version_Minor

This field shall be coded as data type Unsigned8 with the values according to Table 704.

Table 704 – IM_Version_Minor

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Shall be set in this version
0x02 – 0xFF	Reserved

5.2.7.13 Coding of the field IM_Supported

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0: IM_Supported.Profile_specific
This field should be set to zero.

NOTE An application profile can define a different behavior.

Bit 1: IM_Supported.I&M1
This field shall be set according to Table 705.

Table 705 – IM_Supported.I&M1

Value (hexadecimal)	Meaning
0x00	Not supported
0x01	The assigned record allows read and may allow write access.

Bit 2: IM_Supported.I&M2
This field shall be set according to Table 705.

Bit 3: IM_Supported.I&M3
This field shall be set according to Table 705.

Bit 4: IM_Supported.I&M4
This field shall be set according to Table 705.

If used for functional safety, additional rules may apply.

Bit 5: IM_Supported.I&M5
This field shall be set according to Table 705.

Bit 6: IM_Supported.I&M6
This field shall be set according to Table 705.

Bit 7: IM_Supported.I&M7
This field shall be set according to Table 705.

Bit 8: IM_Supported.I&M8
This field shall be set according to Table 705.

Bit 9: IM_Supported.I&M9
This field shall be set according to Table 705.

Bit 10: IM_Supported.I&M10
This field shall be set according to Table 705.

Bit 11: IM_Supported.I&M11
This field shall be set according to Table 705.

Bit 12: IM_Supported.I&M12
This field shall be set according to Table 705.

Bit 13: IM_Supported.I&M13
This field shall be set according to Table 705.

Bit 14: IM_Supported.I&M14
This field shall be set according to Table 705.

Bit 15: IM_Supported.I&M15
This field shall be set according to Table 705.

5.2.7.14 Coding of the field IM_Tag_Function

This field shall be coded as data type VisibleString[32]. The value shall be set by the manufacturer according to 5.2.7.1.

5.2.7.15 Coding of the field IM_Tag_Location

This field shall be coded as data type VisibleString[22]. The value shall be set by the manufacturer according to 5.2.7.1.

5.2.7.16 Coding of the field IM_Date

This field shall be coded as data type VisibleString[16] with the values according to Table 706 or Table 707.

The string format “YYYY-MM-DD’T’HH:MM’Z’” or “YYYY-MM-DD’T’HH:MM’±HH:MM’” is in accordance with ISO 8601-1. The letter ‘T’ indicates that a time value follows after the date and the letter ‘Z’ indicates that the time value is a UTC time.

For legacy reasons the ‘T’ is replaced by “ “ and the time zone information ‘Z’ or ’±HH:MM’ is discarded. Thus the used time zone for Table 706 should be ‘Z’.

Table 706 – IM_Date with time

Octet	Meaning	Usage
0 – 3	YYYY	Year with four digits
4	“–“	Character dash ‘–‘ as separator
5 – 6	MM	Month with two digits
7	“–“	Character dash ‘–‘ as separator
8 – 9	DD	Day with two digits
10	“ “	Character blank ‘ ‘ as separator
11 – 12	HH	Hour with two digits
13	“:”	Character colon ‘:‘ as separator
14 – 15	MM	Minute with two digits

Table 707 – IM_Date without time

Octet	Meaning	Usage
0 – 3	YYYY	Year with four digits
4	“–“	Character dash ‘–‘ as separator
5 – 6	MM	Month with two digits
7	“–“	Character dash ‘–‘ as separator
8 – 9	DD	Day with two digits
10 – 15	“ “	Character blank ‘ ‘ as separator

5.2.7.17 Coding of the field IM_Descriptor

This field shall be coded as data type VisibleString[54]. The value shall be set by the manufacturer according to 5.2.7.1.

5.2.7.18 Coding of the field IM_Signature

This field shall be coded as data type OctetString[54]. The value shall be set by the manufacturer according to 5.2.7.1.

5.2.7.19 Coding of the field IM_Annotation

This field shall be coded as data type UnicodeString8 with a length of 64 octets and values according to Table 708.

All data with type UnicodeString8 shall be left justified. If the text is shorter than the defined string length, the gap shall be filled with blanks.

Table 708 – IM_Annotation

Meaning	Usage
Vendor specific defined Annotation	—

5.2.7.20 Coding of the field IM_OrderID

This field shall be coded as data type UnicodeString8 with a length of 64 octets and values according to Table 709.

All data with type UnicodeString8 shall be left justified. If the text is shorter than the defined string length, the gap shall be filled with blanks.

Table 709 – IM_OrderID

Meaning	Usage
Vendor specific defined Order Identification	—

5.2.7.21 Coding of the field IM_UniqueIdentifier

This field shall be coded as data type UUID with the values according to Table 710.

Table 710 – IM_UniqueIdentifier

Value (hexadecimal)	Meaning
0	Reserved
Other	Manufacturer created unique identifier (UUID) according to ISO/IEC 9834-8. Used as source identification if issuing an event using reporting system or delivering asset management information

5.2.8 Coding section related to Alarm and Diagnosis Data

5.2.8.1 Coding of the field UserStructureIdentifier

This field shall be coded as data type Unsigned16 with the values according to Table 711. This field identifies the structure of the field Data of the AlarmNotification and the structure of the field Data of the alarm data.

Table 711 – UserStructureIdentifier

Value (hexadecimal)	Meaning	Usage
0x0000 – 0x7FFF	ManufacturerSpecific	ManufacturerSpecific Diagnosis shall be used in conjunction with following AlarmTypes: All Diagnosis ASE attached AlarmTypes ManufacturerSpecific Diagnosis shall be used in conjunction with following records: All records containing Diagnosis Data In conjunction with other alarm types, the usage is: Manufacturer specific.
0x8000	ChannelDiagnosis	ChannelDiagnosis shall be used in conjunction with following AlarmTypes: All Diagnosis ASE attached AlarmTypes ChannelDiagnosis shall be used in conjunction with following Records: All records containing Diagnosis Data In conjunction with other alarm types, the usage is: Prohibited
0x8001	Multiple	Reserved

Value (hexadecimal)	Meaning	Usage
0x8002	ExtChannelDiagnosis	ExtChannelDiagnosis shall be used in conjunction with following AlarmTypes: All Diagnosis ASE attached AlarmTypes ExtChannelDiagnosis shall be used in conjunction with following Records: All records containing Diagnosis Data In conjunction with other alarm types, the usage is: Prohibited
0x8003	QualifiedChannel-Diagnosis	QualifiedChannelDiagnosis shall be used in conjunction with following AlarmTypes: All Diagnosis ASE attached AlarmTypes QualifiedChannelDiagnosis shall be used in conjunction with following Records: All records containing Diagnosis Data In conjunction with other alarm types, the usage is: Prohibited
0x8004 – 0x80FF	Reserved	—
0x8100	Maintenance	Maintenance shall be used in conjunction with following AlarmTypes: All Diagnosis ASE attached AlarmTypes Furthermore, the AlarmNotification shall only convey this block if the alarm source contains at least one Maintenance Required entry, or one Maintenance Demanded entry, or one Qualifier_x entry. Otherwise the block shall be omitted. In conjunction with other alarm types, the usage is: Prohibited
0x8101 – 0x81FF	Reserved	—
0x8200	Upload&Retrieval	Upload&Retrieval shall be used in conjunction with upload and retrieval notification. In conjunction with other alarm types, the usage is: Prohibited
0x8201	iParameter	iParameter shall be used in conjunction with upload and retrieval notification. In conjunction with other alarm types, the usage is: Prohibited
0x8202 – 0x82FF	Reserved	—
0x8300	Reporting system RS_LowWatermark	Reporting system shall be used in conjunction with following AlarmTypes: Status alarm In conjunction with other alarm types, the usage is: Prohibited
0x8301	Reporting system RS_Timeout	Reporting system shall be used in conjunction with following AlarmTypes: Status alarm In conjunction with other alarm types, the usage is: Prohibited

Value (hexadecimal)	Meaning	Usage
0x8302	Reporting system RS_Overflow	Reporting system shall be used in conjunction with following AlarmTypes: Status alarm In conjunction with other alarm types, the usage is: Prohibited
0x8303	Reporting system RS_Event	Reporting system shall be used in conjunction with following AlarmTypes: Process alarm In conjunction with other alarm types, the usage is: Prohibited
0x8304 – 0x830F	Reserved	—
0x8310	PE_EnergySavingStatus	Energy saving status shall be used in conjunction with following AlarmTypes: Status alarm In conjunction with other alarm types, the usage is: Prohibited
0x8311 – 0x831F	Reserved	—
0x8320	Channel related Process Alarm reasons ProcessAlarmReason	Channel related Process Alarm reasons shall be used in conjunction with following AlarmTypes: Process alarm In conjunction with other alarm types, the usage is: Prohibited
0x8321 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	—
0xA000 – 0xFFFF	Reserved	—

5.2.8.2 Coding of the field ChannelErrorType

This field shall be coded as data type Unsigned16 with the values according to Table 712, Table 713, Table 714, and Table 715.

Table 712 – ChannelErrorType – range 1

Value (hexadecimal)	Meaning	Assigned text
0x0000	Reserved	Unknown error
0x0001	Short circuit	Short circuit
0x0002	Undervoltage	Undervoltage
0x0003	Overvoltage	Overvoltage
0x0004	Overload	Overload
0x0005	Overtemperature	Overtemperature
0x0006	Wire break	Wire break
0x0007	Upper limit value exceeded	Upper limit value exceeded
0x0008	Lower limit value exceeded	Lower limit value exceeded
0x0009	Error	Error
0x000A	Simulation active	Simulation active

Value (hexadecimal)	Meaning	Assigned text
0x000B	Reserved	Unknown error
0x000C	Reserved	Unknown error
0x000D	Reserved	Unknown error
0x000E	Reserved	Unknown error
0x000F	Default for “parameter missing” [a]	The channel needs (additional) parameters. No or too few parameters are written
0x0010	Default for “parameterization fault” [a]	Parameterization fault. Wrong or too many parameters are written
0x0011	Default for “power supply fault” [a]	Power supply fault
0x0012	Default for “fuse blown / open” [a]	Fuse blown / open
0x0013	Default for “communication fault” [a]	Communication fault. Sequence number wrong / sequence wrong
0x0014	Default for “ground fault” [a]	Ground fault
0x0015	Default for “reference point lost” [a]	Reference point lost
0x0016	Default for “process event lost / sampling error” [a]	Process event lost / sampling error
0x0017	Default for “threshold warning” [a]	Threshold warning
0x0018	Default for “output disabled” [a]	Output disabled
0x0019	Default for “FunctionalSafety event” [a]	FunctionalSafety event
0x001A	Default for “external fault” [a]	External fault
0x001B	Manufacturer specific	Manufacturer specific
0x001C	Manufacturer specific	Manufacturer specific
0x001D	Manufacturer specific	Manufacturer specific
0x001E	Manufacturer specific	Manufacturer specific
0x001F	Default for “temporary fault” [a]	Temporary fault
[a] For legacy devices “Manufacturer specific”		

Table 713 – ChannelErrorType – range 2

Value (hexadecimal)	Meaning	Assigned text
0x0020 – 0x003F	Reserved for common profiles [a]	Central administrative number to unambiguously distinguish between common profiles
0x0040	Functional Safety 0	Mismatch of safety destination address (F_Dest_Add)
0x0041	Functional Safety 1	Safety destination address not valid (F_Dest_Add)
0x0042	Functional Safety 2	Safety source address not valid or mismatch (F_Source_Add)
0x0043	Functional Safety 3	Safety watchdog time value is 0 ms (F_WD_Time, F_WD_Time_2)
0x0044	Functional Safety 4	Parameter “F_SIL” exceeds SIL of specific device application
0x0045	Functional Safety 5	Parameter “F_CRC_Length” does not match the generated values
0x0046	Functional Safety 6	Version of F-Parameter set incorrect

Value (hexadecimal)	Meaning	Assigned text
0x0047	Functional Safety 7	Data inconsistent in received F-Parameter block (CRC1 error)
0x0048	Functional Safety 8	Device specific or unspecified diagnosis information, see manual
0x0049	Functional Safety 9	Save iParameter watchdog time exceeded
0x004A	Functional Safety 10	Restore iParameter watchdog time exceeded
0x004B	Functional Safety 11	Inconsistent iParameters (iParCRC error)
0x004C	Functional Safety 12	F_Block_ID not supported
0x004D	Functional Safety 13	Transmission error: data inconsistent (CRC2 error)
0x004E	Functional Safety 14	Transmission error: timeout (F_WD_Time or F_WD_Time_2 elapsed)
0x004F	Functional Safety 15	Acknowledge needed to enable the channel(s)
0x0050 – 0x005F	Functional Safety 16 to Functional Safety 31	Reserved
0x0060 – 0x00FF	Reserved for common profiles [a]	Central administrative number to unambiguously distinguish between common profiles
[a] Central administrative number to unambiguously distinguish between profiles.		

Table 714 – ChannelErrorType – range 3

Value (hexadecimal)	Meaning	Assigned text
0x0100 – 0x7FFF	Manufacturer specific	Manufacturer specific

Table 715 – ChannelErrorType – range 4

Value (hexadecimal)	Meaning	Assigned text
0x8000	Data transmission impossible	Data Transmission Impossible
0x8001	Remote mismatch	Remote Mismatch
0x8002	Media redundancy mismatch – Ring	Media Redundancy Mismatch
0x8003	Sync mismatch	Sync Mismatch
0x8004	IsochronousMode mismatch	IsochronousMode Mismatch
0x8005	Multicast CR mismatch	Multicast CR Mismatch
0x8006	Reserved	Reserved
0x8007	Fiber optic mismatch	Information for fiber optic links
0x8008	Network component function mismatch	Network functionality problems occur
0x8009	Time mismatch	Time master not existent or precision problems
0x800A	Dynamic frame packing function mismatch	DFP problems occur
0x800B	Media redundancy with planned duplication mismatch	MRPD problems occur

Value (hexadecimal)	Meaning	Assigned text
0x800C	Media redundancy mismatch – Interconnection	Media Redundancy Mismatch
0x800D	Multiple interface mismatch	Information about multiple interface problems
0x800E	Reserved	Unknown error
0x800F	Reserved	Unknown error
0x8010	Power failure over Single Pair Ethernet	Power supply over Single Pair Ethernet reported problems
0x8011 – 0x8FFF	Reserved	Unknown error
0x9000 – 0x9FFF	Reserved for profiles	Profile specific
0xA000 – 0xFFFF	Reserved	Unknown error

5.2.8.3 Coding of the field ChannelNumber

This field shall be coded as data type Unsigned16 with the values according to Table 716.

Table 716 – ChannelNumber

Value (hexadecimal)	Meaning
0x0000 – 0x7FFF	Manufacturer specific
0x8000	Submodule Only one coding for ChannelProperties.Direction shall be used.
0x8001 – 0xFFFF	Reserved

NOTE The ChannelNumber == 0x8000 references the submodule itself.

A distinct channel is addressed by the ChannelNumber in addition to the ChannelProperties.Direction.

Depending on the field ChannelProperties.Accumulative the following meaning shall be applied:

– ChannelProperties.Accumulative == 1, then the field ChannelNumber shall contain the number of the lowest channel of the affected group
– ChannelProperties.Accumulative == 0, then the field ChannelNumber shall contain the number of the affected channel

Only ChannelProperties.Accumulative == 0 shall be used in conjunction with ChannelNumber == 0x8000.

5.2.8.4 Coding of the field ChannelProperties

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: ChannelProperties.Type
This field shall be set according to Table 717.

Table 717 – ChannelProperties.Type

Value (hexadecimal)	Meaning	Usage
0x00	—	Shall be used if the field ChannelNumber contains the value 0x8000 (submodule) Furthermore, it shall be used if none of the types defined below is appropriate.
0x01	1 Bit	The data length of this channel is 1 Bit.
0x02	2 Bit	The data length of this channel is 2 Bit.
0x03	4 Bit	The data length of this channel is 4 Bit.
0x04	8 Bit	The data length of this channel is 8 Bit.
0x05	16 Bit	The data length of this channel is 16 Bit.
0x06	32 Bit	The data length of this channel is 32 Bit.
0x07	64 Bit	The data length of this channel is 64 Bit.
0x08 – 0xFF	Reserved	—

Bit 8: ChannelProperties.Accumulative
This field shall be set according to Table 718.

Table 718 – ChannelProperties.Accumulative

Value (hexadecimal)	Meaning	Usage
0x00	Single	Single channel Diagnosis only for the reported channel
0x01	Accumulative	Multiple channel Accumulative diagnosis from more than one channel

Bit 9 – 10: ChannelProperties.Maintenance
This field shall be set according to Table 719, Table 720 and Table 721.

The dependencies on elements of the field ChannelProperties are specified in Table 720 and the dependencies on AlarmNotification and RecordDataRead(DiagnosisData) are specified in Table 721.

Table 719 – ChannelProperties.Maintenance

MaintenanceRequired Bit 9	MaintenanceDemanded Bit 10:	Usage
0x00	0x00	Fault
0x01	0x00	Maintenance required
0x00	0x01	Maintenance demanded
0x01	0x01	QualifiedChannelQualifier [a]

[a] This indicates that the severity is coded in the field QualifiedChannelQualifier (see Figure 173).

Table 720 – Valid combinations within ChannelProperties

Maintenance	Specifier	Meaning	Valid with
Fault	0x00	All subsequent [a] Fault, MaintenanceRequired, MaintenanceDemanded and QualifiedDiagnosis disappear	ChannelDiagnosis
	0x01	Fault appears	ChannelDiagnosis, ManufacturerSpecificDiagnosis, ExtChannelDiagnosis, QualifiedChannelDiagnosis
	0x02	Fault disappears	
	0x03	Fault disappears but others remain	
Maintenance required	0x00	Reserved	ChannelDiagnosis, ExtChannelDiagnosis, QualifiedChannelDiagnosis
	0x01	MaintenanceRequired appears	
	0x02	MaintenanceRequired disappears	
	0x03	MaintenanceRequired disappears but others remain	
Maintenance demanded	0x00	Reserved	ChannelDiagnosis, ExtChannelDiagnosis, QualifiedChannelDiagnosis
	0x01	MaintenanceDemanded appears	
	0x02	MaintenanceDemanded disappears	
	0x03	MaintenanceDemanded disappears but others remain	
Qualified-Channel-Qualifier	0x00	Reserved	QualifiedChannelDiagnosis
	0x01	QualifiedDiagnosis appears	
	0x02	QualifiedDiagnosis disappears	
	0x03	QualifiedDiagnosis disappears but others remain	

[a] Subsequent means, all ExtChannelErrorTypes for the delivered ChannelErrorType and also the ChannelErrorType itself disappear. Shall be used in AlarmNotification only.

Table 721 – Valid combinations for AlarmNotification and Record-DataRead(DiagnosisData)

Maintenance	Specifier	Meaning	Valid with
Fault	0x00	All subsequent [a] Diagnosis, MaintenanceRequired, MaintenanceDemanded and QualifiedDiagnosis disappear	AlarmNotification
	0x01	Diagnosis appears	AlarmNotification and Record-DataRead(DiagnosisData)
	0x02	Diagnosis disappears	AlarmNotification Special case: "Fieldbus integration of legacy Submodules – Status information" RecordDataRead(DiagnosisData) shall only be used in conjunction with ManufacturerSpecific-Diagnosis
	0x03	Diagnosis disappears but others remain	AlarmNotification
Maintenance required	0x00	Reserved	—
	0x01	MaintenanceRequired appears	AlarmNotification and Record-DataRead(DiagnosisData)
	0x02	MaintenanceRequired disappears	AlarmNotification
	0x03	MaintenanceRequired disappears but others remain	
Maintenance demanded	0x00	Reserved	—
	0x01	MaintenanceDemanded appears	AlarmNotification and Record-DataRead(DiagnosisData)
	0x02	MaintenanceDemanded disappears	AlarmNotification
	0x03	MaintenanceDemanded disappears but others remain	
Qualified diagnosis	0x00	Reserved	—
	0x01	QualifiedDiagnosis appears	AlarmNotification and Record-DataRead(DiagnosisData)
	0x02	QualifiedDiagnosis disappears	AlarmNotification
	0x03	QualifiedDiagnosis disappears but others remain	

[a] Subsequent means, all ExtChannelErrorTypes for the delivered ChannelErrorType and also the ChannelErrorType itself disappear.

Bit 11 – 12: ChannelProperties.Specifier

This field shall be coded with the values according to Table 722. The field ChannelProperties.Specifier is related to the addressed channel and severity (fault or maintenance). The dependencies on elements of the field ChannelProperties are specified in Table 720.

Table 722 – ChannelProperties.Specifier

Value (hexadecimal)	Meaning	Usage
0x00	All subsequent disappear	See Table 720 The Diagnosis ASE contains no longer any entries (of any severity) for this channel
0x01	Appears	An event appears and/or exists further The Diagnosis ASE contains this and possible other entries for this channel.
0x02	Disappears	An event disappears and/or exists no longer The Diagnosis ASE contains no longer any entries of the same severity for this channel
0x03	Disappears but others remain	An event disappears The Diagnosis ASE still contains other entries of the same severity for this channel

NOTE The conveyed data of the AlarmNotification depends on the creation time and can differ from the actual content of the Diagnosis ASE. The AlarmNotification is used to convey the changes and the Diagnosis ASE to contain the actual status.

Bit 13 – 15: ChannelProperties.Direction
This field shall be set according to Table 723.

Table 723 – ChannelProperties.Direction

Value (hexadecimal)	Meaning
0x00	Manufacturer specific
0x01	Input
0x02	Output
0x03	Input/Output
0x04 – 0x07	Reserved

5.2.8.5 Coding of the field ExtChannelErrorType

5.2.8.5.1 General

This field shall be coded as data type Unsigned16. The value of this field depends on the field ChannelErrorType. The values shall be set according to Table 724 and Table 725.

Table 724 – ExtChannelErrorType

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Definition depending on the ChannelErrorType (see tables below).

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 726, Table 727, Table 728, Table 729, Table 730, Table 731, Table 732, Table 733, Table 734, Table 735, Table 736, Table 737, Table 738, Table 739 and Table 740.

Table 725 – Allowed combinations of ChannelErrorType, ExtChannelErrorType, and ExtChannelAddValue

ChannelErrorType [Number range]	ExtChannelErrorType [Number range]	ExtChannelAddValue [Number range]	Usage
Normative	Normative	Normative	Allowed
Normative	Vendor specific	Vendor specific	Allowed
Normative	Profile specific	Profile specific	Allowed
Profile specific	Profile specific	Profile specific	Allowed
Profile specific	Vendor specific	Vendor specific	Allowed
Profile specific	Normative	Normative	Prohibited
Vendor specific	Vendor specific	Vendor specific	Allowed
Vendor specific	Normative	Normative	Prohibited
Vendor specific	Profile specific	Profile specific	Prohibited

5.2.8.5.2 ExtChannelErrorType for ChannelErrorType 0 – 0xFF

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 726 and Table 727.

Table 726 – ExtChannelErrorType for ChannelErrorType 0 – 0xFF

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Accumulative Info	Alarm/diagnosis
0x8001 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

Table 727 – Additional ExtChannelErrorType for ChannelErrorType 0x0F and 0x10

Value (hexadecimal)	Meaning	Usage
0x8001	Parameter fault detail	Alarm/diagnosis
0x8002	Consistency fault detail	Alarm/diagnosis

5.2.8.5.3 ExtChannelErrorType for ChannelErrorType 0x0100 – 0x7FFF

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 728.

Table 728 – ExtChannelErrorType for ChannelErrorType 0x0100 – 0x7FFF

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Accumulative Info	Alarm/diagnosis
0x8001 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

5.2.8.5.4 ExtChannelErrorType for ChannelErrorType "Data transmission impossible"

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 729.

Table 729 – ExtChannelErrorType for ChannelErrorType "Data transmission impossible"

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Link State mismatch – Link down	Alarm/diagnosis
0x8001	MAUType mismatch	Alarm/diagnosis
0x8002	Line Delay mismatch	Alarm/diagnosis
0x8003 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

5.2.8.5.5 ExtChannelErrorType for ChannelErrorType "Remote mismatch"

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 730.

Table 730 – ExtChannelErrorType for ChannelErrorType “Remote mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Peer name of station mismatch	Alarm/diagnosis
0x8001	Peer name of port mismatch	Alarm/diagnosis
0x8002	Peer RT_CLASS_3 mismatch [a]	Alarm/diagnosis
0x8003	Peer MAUType mismatch	Alarm/diagnosis
0x8004	Peer MRP domain mismatch	Alarm/diagnosis
0x8005	No peer detected	Alarm/diagnosis
0x8006	Reserved	—
0x8007	Peer Line Delay mismatch	Alarm/diagnosis
0x8008	Peer PTCP mismatch [b]	Alarm/diagnosis
0x8009	Peer Preamble Length mismatch	Alarm/diagnosis
0x800A	Peer Fragmentation mismatch	Alarm/diagnosis
0x800B	Peer MRP Interconnection domain mismatch	Alarm/diagnosis
0x800C – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

[a] This event will occur, if the remote RTClass3_PortStatus.State is not RTCLASS3_RUN. This can also happen, if the remote IRDATAUUID is not equal to the local IRDATAUUID.

[b] This event will occur, if the remote PTCP_SubdomainUUID is equal to the local PTCP_SubdomainUUID and the remote PTCP_MasterSourceAddress is not equal to the local PTCP_MasterSourceAddress.

5.2.8.5.6 ExtChannelErrorType for ChannelErrorType “Media redundancy mismatch – Ring” and ExtChannelErrorType for ChannelErrorType “Media redundancy mismatch – Interconnection”

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 731.

Table 731 – ExtChannelErrorType for ChannelErrorType “Media redundancy mismatch – Ring”

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Manager role fail	Alarm/diagnosis
0x8001	MRP ring open	Alarm/diagnosis
0x8002	Reserved	—
0x8003	Multiple manager	Alarm/diagnosis
0x8004 – 0x800F	Reserved	—
0x8010 – 0x87EF	Reserved for “Multiple MRP rings”	See Formula (64)
0x87F0 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

The source of this diagnosis shall be the SlotNumber/SubslotNumber interface hosting the used "ring" ports.

If "PDInterfaceMrpDataAdjust with BlockVersionLow = 1" is used, the ExtChannelErrorType shall integrate MRP_Instance according to Formula (64).

$$\text{ExtChannelErrorType} = 0x800X + (\text{MRP\_Instance} \times 0x10) \quad (64)$$

where

ExtChannelErrorType	is the ExtChannelErrorType used for the signaling of an event
0x800X	is the base value of the ExtChannelErrorType from Table 731
MRP_Instance	is the MRP_Instance of the MRP ring issuing the event
0x10	is a constant value

Table 732 – ExtChannelErrorType for ChannelErrorType "Media redundancy mismatch – Interconnection"

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	MIM role fail	Alarm/diagnosis
0x8001	MIM reconfiguration rejected	Alarm/diagnosis
0x8002	MRP Interconnection open, secondary link	Alarm/diagnosis
0x8003	MIC missing, secondary link	Alarm/diagnosis
0x8004	MRP Interconnection open, primary link	Alarm/diagnosis
0x8005	MIC missing, primary link	Alarm/diagnosis
0x8006	Multiple MIMs	Alarm/diagnosis
0x8007	Too many MICs	Alarm/diagnosis
0x8008 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

The source of this diagnosis shall be the SlotNumber/SubslotNumber representing the signaling "interconnection" port.

5.2.8.5.7 ExtChannelErrorType for ChannelErrorType "Sync mismatch" and for ChannelErrorType "Time mismatch"

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 733.

Table 733 – ExtChannelErrorType for ChannelErrorType "Sync mismatch" and for ChannelErrorType "Time mismatch"

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	No sync message received	Alarm/diagnosis
0x8001 – 0x8002	Reserved	—
0x8003	Jitter out of boundary	Alarm/diagnosis
0x8004 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

5.2.8.5.8 ExtChannelErrorType for ChannelErrorType "Isochronous mode mismatch"

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 734.

Table 734 – ExtChannelErrorType for ChannelErrorType "Isochronous mode mismatch"

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Output Time Failure – Output update missing or out of order	Alarm/diagnosis
0x8001	Input Time Failure	Alarm/diagnosis
0x8002	Master Life Sign Failure – Error in Master Life Sign update detected	Alarm/diagnosis
0x8003 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

5.2.8.5.9 ExtChannelErrorType for ChannelErrorType "Multicast CR mismatch"

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 735.

Table 735 – ExtChannelErrorType for ChannelErrorType "Multicast CR mismatch"

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Multicast Consumer CR timed out	Alarm/diagnosis
0x8001	Address resolution failed	Alarm/diagnosis
0x8002 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

5.2.8.5.10 ExtChannelErrorType for ChannelErrorType “Fiber optic mismatch”

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 736.

Table 736 – ExtChannelErrorType for ChannelErrorType “Fiber optic mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Power Budget	Alarm/diagnosis
0x8001	SFP – Temperature threshold violation (High)	Alarm/diagnosis
0x8002	SFP – TX Bias threshold violation (High)	Alarm/diagnosis
0x8003	SFP – TX Bias threshold violation (Low)	Alarm/diagnosis
0x8004	SFP – TX Power threshold violation (High)	Alarm/diagnosis
0x8005	SFP – TX Power threshold violation (Low)	Alarm/diagnosis
0x8006	SFP – RX Power threshold violation (High)	Alarm/diagnosis
0x8007	SFP – RX Power threshold violation (Low)	Alarm/diagnosis
0x8008	SFP – TX Fault State indication	Alarm/diagnosis
0x8009	SFP – RX Loss State indication	Alarm/diagnosis
0x800A – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

5.2.8.5.11 ExtChannelErrorType for ChannelErrorType “Network component function mismatch”

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 737.

Table 737 – ExtChannelErrorType for ChannelErrorType “Network component function mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Frame dropped – no resource	Alarm/diagnosis
0x8001 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

5.2.8.5.12 ExtChannelErrorType for ChannelErrorType “Dynamic Frame Packing function mismatch”

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 738.

Table 738 – ExtChannelErrorType for ChannelErrorType "Dynamic Frame Packing function mismatch"

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000 – 0x80FF	Reserved	—
0x8100	Frame late error for FrameID "0x0100"	Alarm/diagnosis
0x8101 – 0x8FFE	See Formula (65)	Alarm/diagnosis
0x8FFF	Frame late error for FrameID "0x0FFF"	Alarm/diagnosis
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

The source of a "Dynamic Frame Packing function mismatch" is the interface submodule using the DFP packing function.

The used ExtChannelErrorType shall be calculated according to Formula (65).

$$\text{ExtChannelErrorType} = 0x8000 + \text{FrameID} \quad (65)$$

where

ExtChannelErrorType is the ExtChannelErrorType used for the signaling of an event

0x8000 is the base value of the ChannelNumber

FrameID is the FrameID of the frame issuing the event

5.2.8.5.13 ExtChannelErrorType for ChannelErrorType "Media redundancy with planned duplication mismatch"

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 739.

Table 739 – ExtChannelErrorType for ChannelErrorType "Media redundancy with planned duplication mismatch"

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000 – 0x86FF	Reserved	—
0x8700	MRPD duplication void for FrameID "0x0700"	Alarm/diagnosis
0x8701 – 0x8FFE	See Formula (65)	Alarm/diagnosis
0x8FFF	MRPD duplication void for FrameID "0x0FFF"	Alarm/diagnosis
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

The source of a "Media redundancy with planned duplication mismatch" is the interface submodule using the MRPD function.

NOTE Searching for the smallest SlotNumber and then for the smallest SubslotNumber of this slot identifies the lowest SlotNumber/SubslotNumber combination.

The used ExtChannelErrorType shall be calculated according to Formula (65).

5.2.8.5.14 ExtChannelErrorType for ChannelErrorType "Multiple interface mismatch"

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 740.

Table 740 – ExtChannelErrorType for ChannelErrorType "Multiple interface mismatch"

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	StandardGateway mismatch	Alarm/diagnosis
0x8001	NameOfStation of the interface is not unambiguous	Alarm/diagnosis
0x8002	IPAddress range of the interface is not unambiguous	Alarm/diagnosis
0x8003	Conflicting MultipleInterfaceMode.NameOfDevice mode	Alarm/diagnosis
0x8004	StandardGateway inactive	Alarm/diagnosis
0x8005 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

5.2.8.5.15 ExtChannelErrorType for ChannelErrorType "Power failure over Single Pair Ethernet"

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 741.

Table 741 – ExtChannelErrorType for ChannelErrorType "Power failure over Single Pair Ethernet"

Value (hexadecimal)	Meaning	Usage
0x0000	Unspecified	Alarm/diagnosis
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	SPE power supply – Short circuit	Alarm/diagnosis
0x8001	SPE power supply – Open circuit	Alarm/diagnosis
0x8002	SPE power supply – Voltage level	Alarm/diagnosis
0x8003	SPE power supply – Current level	Alarm/diagnosis
0x8004 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

5.2.8.6 Coding of the field ExtChannelAddValue

5.2.8.6.1 General

This field shall be coded as data type Unsigned32 with the values according to Table 742.

Table 742 – Values for ExtChannelAddValue

Value (hexadecimal)	Meaning
0x00000000	Definition depending on ChannelError and ExtChannelErrorType Default for "No additional information"
0x00000001 – 0xFFFFFFFF	Definition depending on ChannelError and ExtChannelErrorType

5.2.8.6.2 Coding of the field ExtChannelAddValue for ChannelErrorType := 0 – 0x7FFF

5.2.8.6.2.1 ExtChannelErrorType := 0x8000

This field shall be coded with the values according to Table 743 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0 – 0x7FFF.

Table 743 – Values for "Accumulative Info"

Bitposition	Value (hexadecimal)	Meaning
Bit 0	0x00	ChannelNumber is not affected
	0x01	ChannelNumber is affected
Bit 1	0x00	ChannelNumber + 1 is not affected
	0x01	ChannelNumber + 1 is affected
Bit 2	0x00	ChannelNumber + 2 is not affected
	0x01	ChannelNumber + 2 is affected
…	…	…
Bit30	0x00	ChannelNumber + 30 is not affected
	0x01	ChannelNumber + 30 is affected
Bit31	0x00	ChannelNumber + 31 is not affected
	0x01	ChannelNumber + 31 is affected

5.2.8.6.2.2 ExtChannelErrorType := 0x8001

This field shall be coded with the values according to Table 744, Table 745, and Table 746 if the field ExtChannelErrorType contains the value 0x8001 and the field ChannelErrorType contains the value 0 – 0xFF.

Table 744 – Values for ExtChannelErrorType "Parameter fault detail"

Value (hexadecimal)	Meaning	Usage
Bit 31 – 16 (Unsigned16)	ExtChannelAddValue.Index	Index of a record
Bit 15 – 0 (Unsigned16)	ExtChannelAddValue.Offset	Offset in this record

Table 745 – Values for ExtChannelAddValue.Index

Value (hexadecimal)	Meaning	Usage
0x0000 – 0xFFFE	Index of the record being the reason of the fault	—
0xFFFF	Unknown index	Should only be used for ChannelErrorType "Parameterization fault" in conjunction with "Unknown offset"

Table 746 – Values for ExtChannelAddValue.Offset

Value (hexadecimal)	Meaning	Usage
0x0000 – 0xFFFF	Offset inside the field RecordDataWrite of the record referenced by ExtChannelAddValue.Index Special case "0x0000": Additionally used for "Unknown offset"	Special case "0x0000": Should only be used for ChannelErrorType "Parameter missing" or for ChannelErrorType "Parameterization fault" in conjunction with "Consistency fault"

5.2.8.6.2.3 ExtChannelErrorType := 0x8002

This field shall be coded with the values according to Table 747, Table 748, and Table 746 if the field ExtChannelErrorType contains the value 0x8002 and the field ChannelErrorType contains the value 0 – 0xFF.

Table 747 – Values for ExtChannelErrorType "Consistency fault detail"

Value (hexadecimal)	Meaning	Usage
Bit 31 – 16 (Unsigned16)	ExtChannelAddValue.Index	Index of a record
Bit 15 – 0 (Unsigned16)	ExtChannelAddValue.Offset	Offset in this record

Table 748 – Values for ExtChannelAddValue.Index

Value (hexadecimal)	Meaning	Usage
0x0000 – 0xFFFE	Index of the record being the reason of the fault	—
0xFFFF	Consistency fault More than one record involved.	Should only be used for ChannelErrorType “Parameterization fault” in conjunction with “Unknown offset”

5.2.8.6.3 Coding of the field ExtChannelAddValue for ChannelErrorType “Fiber optic mismatch”

5.2.8.6.3.1 Power Budget

This field shall be coded with the values according to Table 749 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0x8007.

Table 749 – Values for “Fiber optic mismatch” – “Power Budget”

Value (hexadecimal)	Meaning
0x00 – 0x3E7	PowerBudget in 0,1 dB steps [0..99,9 dB]
0x03E8 – 0xFFFFFFFE	Reserved
0xFFFFFFFF	Unknown

5.2.8.6.4 Coding of the field ExtChannelAddValue for ChannelErrorType “Network component function mismatch”

5.2.8.6.4.1 Frame dropped

This field shall be coded with the values according to Table 750 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0x8008.

Table 750 – Values for “Network component function mismatch” – “Frame dropped”

Value (hexadecimal)	Meaning
0x00000000 – 0x00FFFFFF	Number of dropped frames in case of no resource
0x01000000 – 0xFFFFFFFF	Reserved

5.2.8.6.5 Coding of the field ExtChannelAddValue for ChannelErrorType “Remote mismatch”

5.2.8.6.5.1 Peer CableDelay mismatch

This field shall be coded with the values according to Table 751 if the field ExtChannelErrorType contains the value 0x8007 and the field ChannelErrorType contains the value 0x8001.

Table 751 – Values for "Remote mismatch" – "Peer CableDelay mismatch"

Value (hexadecimal)	Meaning	Usage
0x00 – 0x32	Error of measurement	No signaling
0x33 – 0x3B9ACA00	CableDelay difference between peers	Signal deviation
0x3B9ACA01 – 0xFFFFFFFF	Reserved	—

5.2.8.6.6 Coding of the field ExtChannelAddValue for ChannelErrorType "Multiple interface mismatch"

5.2.8.6.6.1 Conflicting MultipleInterfaceMode.NameOfDevice mode

This field shall be coded with the values according to Table 752 if the field ExtChannelErrorType contains the value 0x8003 and the field ChannelErrorType contains the value 0x800D.

Table 752 – Values for "Multiple interface mismatch" – "Conflicting MultipleInterfaceMode.NameOfDevice mode"

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
0x01	MultipleInterfaceMode.NameOfDevice for this interface is set to zero but one other interface is set to 1.	Inconsistency
0x02	MultipleInterfaceMode.NameOfDevice for this interface is set to 1 but one other interface is set to zero.	Inconsistency
Other	Reserved	—

5.2.8.6.6.2 Inactive StandardGateway

This field shall be coded with the values according to Table 753 if the field ExtChannelErrorType contains the value 0x8004 and the field ChannelErrorType contains the value 0x800D.

Table 753 – Values for "Multiple interface mismatch" – "Inactive StandardGateway"

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
Other	StandardGateway	The inactive StandardGateway which is the reason for this diagnosis.

5.2.8.7 Coding of the field QualifiedChannelQualifier

This field shall be coded as data type Unsigned32. The values shall be set according to Table 754. There shall be set only none or one bit in one QualifiedChannelDiagnosis.

The QualifiedChannelQualifier shall be set to zero if the QualifiedChannelDiagnosis format is used in conjunction with severity Fault, MaintenanceRequired or MaintenanceDemanded.

Table 754 – Values for QualifiedChannelQualifier

Bitposition	Value (hexadecimal)	Meaning	Usage
Bit 0 – 2	0x00	—	Reserved
Bit 3	0x00	Qualifier_3 not set	Defined/summarized as Advice
	0x01	Qualifier_3 is set	
…	…	…	
Bit 6	0x00	Qualifier_6 not set	
	0x01	Qualifier_6 is set	
Bit 7	0x00	Qualifier_7 not set	Defined/summarized as MaintenanceRequired
	0x01	Qualifier_7 is set	
…	…	…	
Bit 16	0x00	Qualifier_16 not set	
	0x01	Qualifier_16 is set	
Bit 17	0x00	Qualifier_17 not set	Defined/summarized as MaintenanceDemanded
	0x01	Qualifier_17 is set	
…	…	…	
Bit 26	0x00	Qualifier_26 not set	
	0x01	Qualifier_26 is set	
Bit 27	0x00	Qualifier_27 not set	Defined/summarized as Fault
	0x01	Qualifier_27 is set	
…	…	…	
Bit 31	0x00	Qualifier_31 not set	
	0x01	Qualifier_31 is set	

5.2.8.8 Coding of the field MaintenanceStatus

This field shall be coded as data type Unsigned32. The values shall be set according to Table 755. At least one of the bits below shall be set to 1 in order to convey this alarm block. Otherwise this block shall be omitted.

Table 755 – Values for MaintenanceStatus

Bitposition	Bit name	Value (hexadecimal)	Meaning
Bit 0	MaintenanceRequired [a]	0x00	No maintenance required information in the Diagnosis ASE available for the reporting submodule
		0x01	Maintenance required information available
Bit 1	MaintenanceDemanded [b]	0x00	No maintenance demanded information in the Diagnosis ASE available for the reporting submodule
		0x01	Maintenance demanded information available
Bit 2	Reserved	0x00	Reserved
Bit 3	Qualifier_3	0x00	No information in the Diagnosis ASE available for the reporting submodule
		0x01	Information available

Bitposition	Bit name	Value (hexadecimal)	Meaning
…	—	—	…
Bit 30	Qualifier_30	0x00	No information in the Diagnosis ASE available for the reporting submodule
		0x01	Information available
Bit 31	Qualifier_31	0x00	No information in the Diagnosis ASE available for the reporting submodule
		0x01	Information available
[a] This bit summarizes the MaintenanceRequired information in all codings. See Table 754.			
[b] This bit summarizes the MaintenanceDemanded information in all codings. See Table 754.			

The classification of fault, maintenance and normal operation according to their severity is shown in Figure 173.

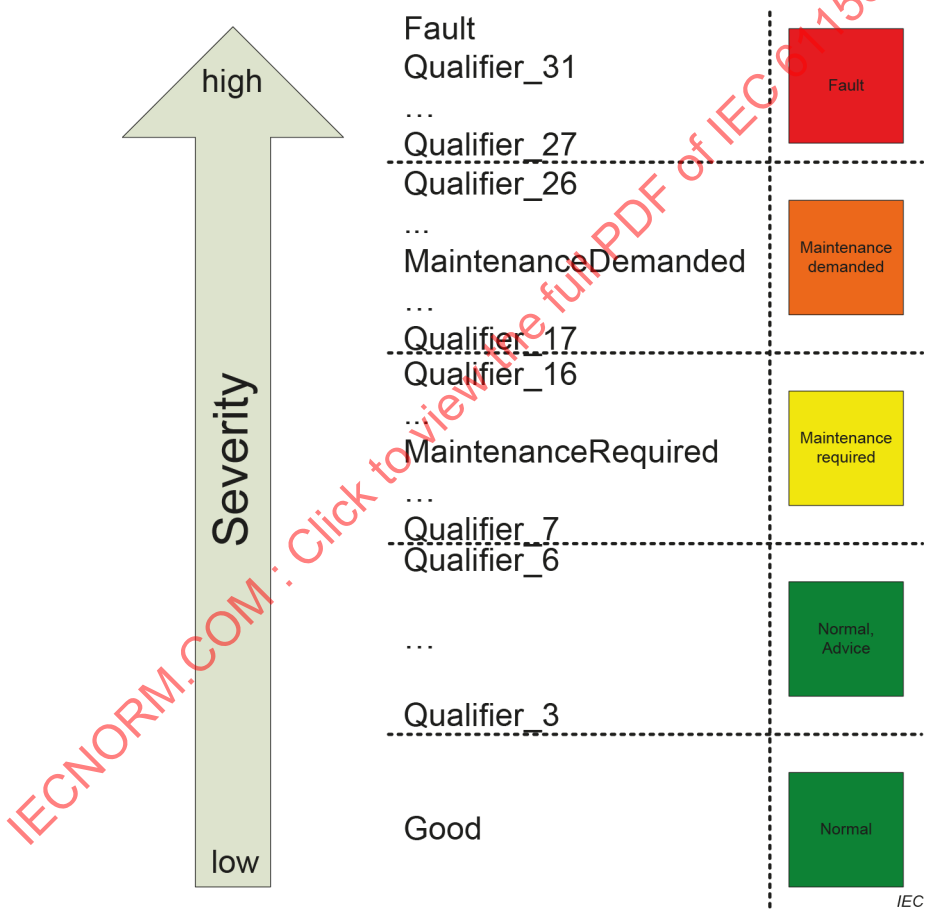


Figure 173 – Severity classification of fault, maintenance and normal operation

5.2.8.9 Coding of the field ManufacturerData

The data types defined in IEC 61158-5-10, Clause 5 shall be used for ManufacturerData.

The maximum length of this field is bound to the transport via the alarm channel. Thus, MaxAlarmDataLength limits the amount of ManufacturerData.

5.2.9 Coding section related to upload and retrieval

5.2.9.1 Coding of the field URRecordIndex

This field shall be coded as data type Unsigned32 with the values according to Table 756.

Table 756 – URRecordIndex

Value (hexadecimal)	Meaning
0x00000000 – 0x0000FFFF	Index of the used record
0x00010000 – 0xFFFFFFFF	Reserved

5.2.9.2 Coding of the field URRecordLength

This field shall be coded as data type Unsigned32 with the values according to Table 757.

Table 757 – URRecordLength

Value (hexadecimal)	Meaning
0x00000000	Write stored records to the IO device.
0x00000001 – 0xFFFFFFFF	Write stored records to the IO device, if the length of each selected record is less than or equal to URRecordLength.

5.2.10 Coding section related to iParameter

5.2.10.1 Coding of the field iPar_Req_Header

This field shall be coded as data type Unsigned32 with the values according to Table 758.

Table 758 – iPar_Req_Header

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Depending on the used application protocol.

5.2.10.2 Coding of the field Max_Segm_Size

This field shall be coded as data type Unsigned32 with the values according to Table 759.

Table 759 – Max_Segm_Size

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Depending on the used application protocol.

5.2.10.3 Coding of the field Transfer_Index

This field shall be coded as data type Unsigned32 with the values according to Table 760.

Table 760 – Transfer_Index

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Depending on the used application protocol.

5.2.10.4 Coding of the field Total_iPar_Size

This field shall be coded as data type Unsigned32 with the values according to Table 761.

Table 761 – Total_iPar_Size

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Depending on the used application protocol.

5.2.11 Coding section related to NME

5.2.11.1 Coding of the field NMEDomainNameLength

This field shall be coded as data type Unsigned16.

5.2.11.2 Coding of the field NMEDomainName

Each device shall be equipped with a NMEDomainName which is unique for this NME domain.

The default value is an empty string which indicates that no NMEDomainName is assigned.

This field shall be coded as data type OctetString with 0 to 240 octets according to 4.3.1.4.16.

5.2.11.3 Coding of the field NMEDomainUUID

This field shall be coded as data type UUID with the values according to Table 762.

The NMEDomainUUID shall be calculated according to Formula (66).

$$\text{NMEDomainUUID} = \text{MD5}(\text{NMEDomainName}) \quad (66)$$

where

NMEDomainUUID is the NME domain UUID

MD5 is the IETF RFC 6151 defined function to create a 128 bit fingerprint

NMEDomainName is the variable length input parameter NME domain name

Table 762 – NMEDomainUUID

Value (UUID)	Meaning
00000000-0000-0000-0000-000000000000	No NME domain assigned
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF	UUID identifying a NME domain using SNMP / LLDP/ DCP

5.2.11.4 Coding of the field NMENameLength

This field shall be coded as data type Unsigned16.

5.2.11.5 Coding of the field NMEName

Each NME shall be equipped with a NMEName which is unique.

The default value is an empty string which indicates that no NMEName is assigned.

This field shall be coded as data type OctetString with 0 to 240 octets according to 4.3.1.4.16.

5.2.11.6 Coding of the field NMENameUUID

This field shall be coded as data type UUID with the values according to Table 763.

The NMENameUUID shall be calculated according to Formula (67).

$$NMENameUUID = MD5(NMEName) \quad (67)$$

where

NMENameUUID is the NME UUID

MD5 is the IETF RFC 6151 defined function to create a 128 bit fingerprint

NMEName is the variable length input parameter NME name

Table 763 – NMENameUUID

Value (UUID)	Meaning
00000000-0000-0000-0000-000000000000	No NME assigned
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF	UUID identifying an NME using SNMP / LLDP / DCP

5.2.11.7 Coding of the field NMEParameterUUID

This field shall be coded as data type UUID with the values according to Table 764.

The NMEParameterUUID shall reflect the parameterization value.

Table 764 – NMEParameterUUID

Value (UUID)	Meaning
00000000-0000-0000-0000-000000000000	Unconfigured
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF	UUID identifying an NME parameter set within the NME domain.

5.2.11.8 Coding of the field NMENameAddressSubtype

This field shall be coded as data type Unsigned8 with the values according to Table 765.

Table 765 – NMENameAddressSubtype

Value (hexadecimal)	Meaning
0x01	The management address of the NME is coded as IPv4 address.
0x06	The management address of the NME is coded as MAC address.
Other	Reserved

5.2.11.9 Coding section related to Physical Query Path Data

5.2.11.9.1 Coding of the field StreamIdentification

This field shall be coded as data type Unsigned64 with the values according to Table 766.

Table 766 – StreamIdentification

Value (hexadecimal)	Meaning
0x00	Unknown Used for UNIAddStreamReq when no stream identifier is known for this stream by the requestor
Other	NME provided stream identifier Used by UNIAddStreamReq, UNIRenewStreamReq, UNIRemoveStreamReq to reference a stream

5.2.11.9.2 Coding of the field EndpointStationName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.16.

5.2.11.9.3 Coding of the field StreamControl

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: StreamControl.Priority

The bit shall be coded with the values according to Table 767.

Table 767 – StreamControl.Priority

Value (hexadecimal)	Meaning
0x00	Time-aware stream, class HIGH
0x01	Time-aware stream, class LOW
0x02	Stream, class RT
0x03	Reserved

Bit 2: StreamControl.Redundancy

The bit shall be coded with the values according to Table 768.

Table 768 – StreamControl.Redundancy

Value (hexadecimal)	Meaning
0x00	Non-redundant stream
0x01	Redundant stream

Bit 3: StreamControl.Append

The bit shall be coded with the values according to Table 769.

Table 769 – StreamControl.Append

Value (hexadecimal)	Meaning
0x00	New stream is requested
0x01	Append endpoint to an existing stream

Bit 4: StreamControl.Dependency

The bit shall be coded with the values according to Table 770.

Table 770 – StreamControl.Dependency

Value (hexadecimal)	Meaning
0x0	Mandatory stream, do not establish any other stream of this request if this stream cannot be established
0x1	Optional stream, establish other streams of this request even if this stream cannot be established

Bit 5 – 15: StreamControl.Reserved

The bits shall be coded with the value zero.

5.2.11.9.4 Coding of the field UpdateInterval

This field shall be coded as data type Unsigned16 with values according to Table 771 and Figure 174. The time base shall be 31,25 µs.

The UpdateInterval is defined by the used SendClock and Reduction Ratio. It is the upper bound for the MaxCalculatedLatency.

Table 771 – Values of UpdateInterval

Value (decimal)	LengthOfPeriod	Meaning
0	—	Reserved
1	31,25 µs	Optional
2 – 31	—	Optional
32	1 ms	Optional
33 – 63	—	Optional
64	2 ms	Optional
65 – 127	—	Optional
128	4 ms	Optional
129 – 4 095	—	Optional
4 096	128 ms	Optional
4 097 – 31 999	—	Optional
32 000	1 s	Optional
32 001 – 63 999	—	Optional
64 000	2 s	Optional
64 001 – 65 535	—	Optional

The value of the LengthOfUpdateInterval shall be calculated according to Formula (68).

$$\text{LengthOfUpdateInterval} = \text{UpdateInterval} \times 31{,}25\ \mu\text{s} \quad (68)$$

where

LengthOfUpdateInterval is the length of the update period

UpdateInterval is a factor derived from the send clock factor definition

Figure 174 shows the maximum, the minimum and the selected UpdateInterval.

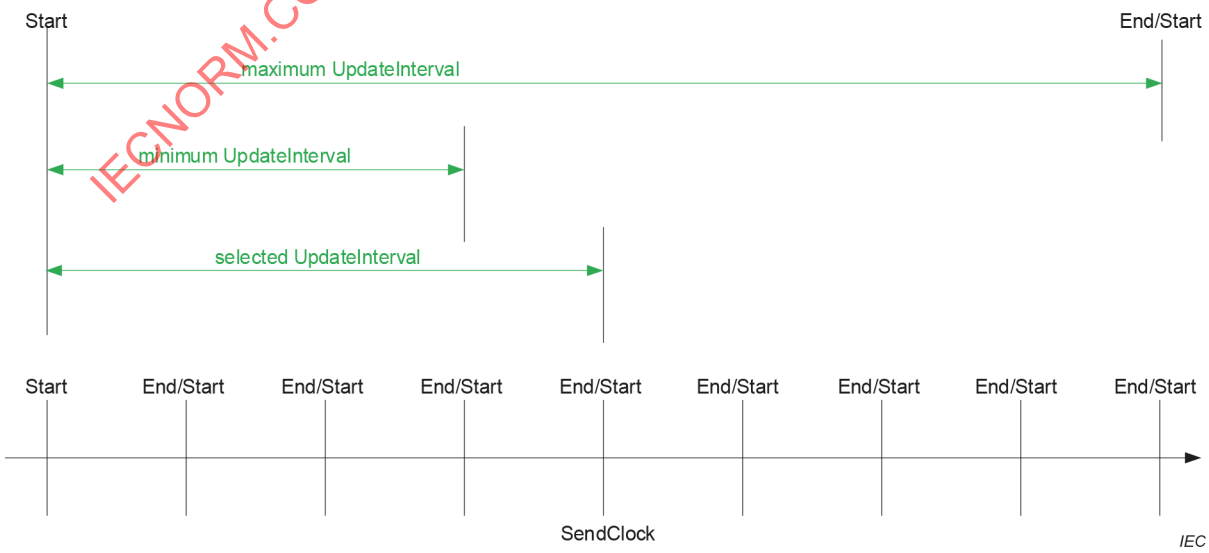


Figure 174 – UpdateInterval measurement

5.2.11.9.5 Coding of the field NetworkDeadline

This field shall be coded as data type Unsigned32 according to Table 772 and Figure 175.

The NetworkDeadline is a point in time defined for each SendClock. It defines the latest allowed arrival time at the consumer for frames to which this deadline applies. It applies to “Time-aware stream, class HIGH” only.

Table 772 – NetworkDeadline

Value (hexadecimal)	Meaning
0x00000000	No deadline
0x00000001 – 0xFFFFFFFF	The deadline in microseconds

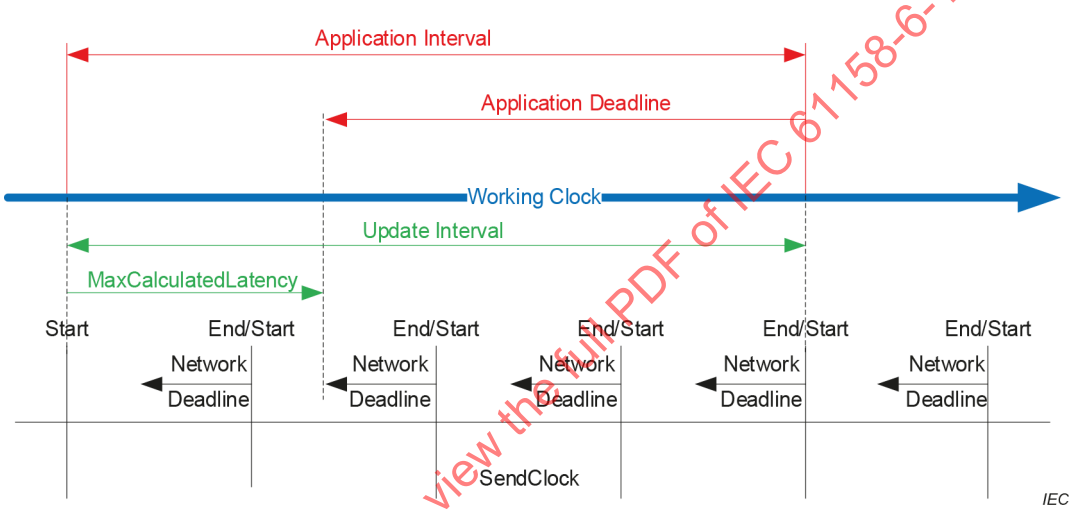


Figure 175 – Deadline measurement

5.2.11.9.6 Coding of the field ApplicationDeadline

This field shall be coded as data type Unsigned32 according to Table 773, Table 774, and Figure 175.

The ApplicationDeadline is a point in time defined for both the associated Application Interval, coded as TimeDataCycle, and the UpdateInterval.

It defines the latest allowed arrival time at the consumer for frames to which this deadline applies.

Table 773 – Application Interval

Value	Meaning
25 µs – 32 ms	Possible Application Interval

Table 774 – ApplicationDeadline

Value (hexadecimal)	Meaning
0x00000000	No deadline
0x00000001 – 0x00007D00	The deadline in microseconds The value shall be less than the Application Interval.
0x00007D01 – 0xFFFFFFFF	Reserved

5.2.11.9.7 Coding of the field PduSize

This field shall be coded as data type Unsigned16 according to Table 775.

Table 775 – PduSize

Value (hexadecimal)	Meaning
0x0000 – 0x0039	Reserved
0x0040 – 0x05F2	The size of DLPDUs in octets according to IEEE Std 802.3 (excluding Preamble, Start Frame Delimiter). For bandwidth calculation IPG, Preamble and Start Frame Delimiter shall be added.
0x0040 – 0x0800	If envelope frames are used. For bandwidth calculation IPG, Preamble and Start Frame Delimiter shall be added.
Other	Reserved

5.2.11.9.8 Coding of the field StreamTCI

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 11: StreamTCI.VID

The bits shall be coded with the values according to Table 776.

Table 776 – StreamTCI.VID

Value	Meaning
0	Reserved
1 – 4 095	TCI.VID used for the stream

Bit 12 – 14: StreamTCI.PCP

The bits shall be coded with the values according to Table 777.

Table 777 – StreamTCI.PCP

Value	Meaning
0 – 7	TCI.PCP used for the stream

Bit 15: StreamTCI.Reserved

The bit shall be coded with the value zero.

5.2.11.9.9 Coding of the field MaxCalculatedLatency

This field shall be coded as data type Unsigned64 according to Table 778, Figure 176, Figure 177, and Figure 178.

MaxCalculatedLatency contains the maximum transmission delay for “Time-aware stream, class HIGH”, only. The timespan starts with the begin of the phase in which a frame is injected by a PPM and ends with the latest point in time in which it is received at the corresponding CPM.

Table 778 – MaxCalculatedLatency

Value (hexadecimal)	Meaning
0x00000000	No Latency can be calculated
0x00000001 – 0xFFFFFFFF	The assured latency, from application to application, in nanoseconds for this stream.
Other	Reserved

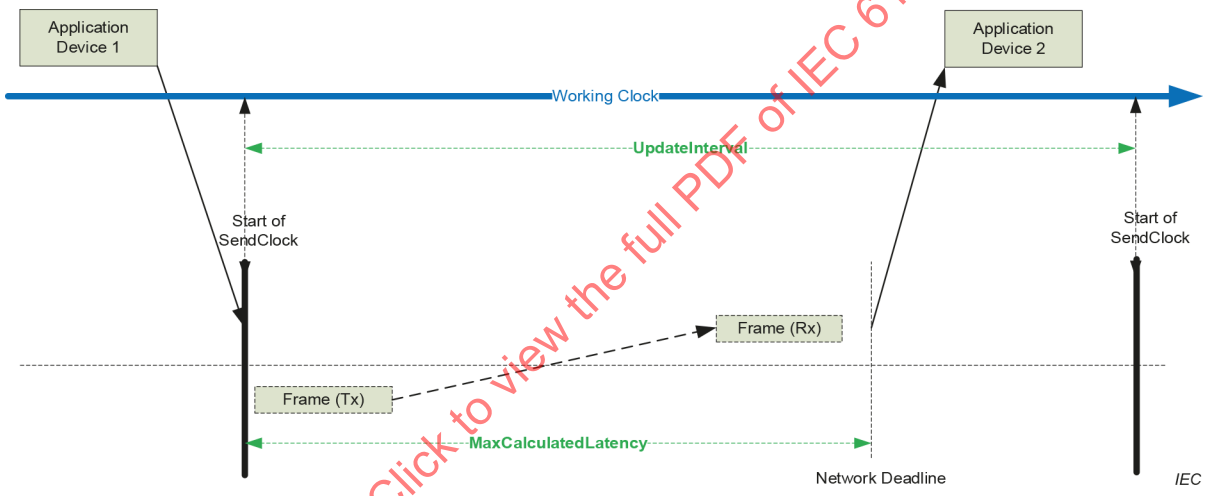


Figure 176 – MaxCalculatedLatency

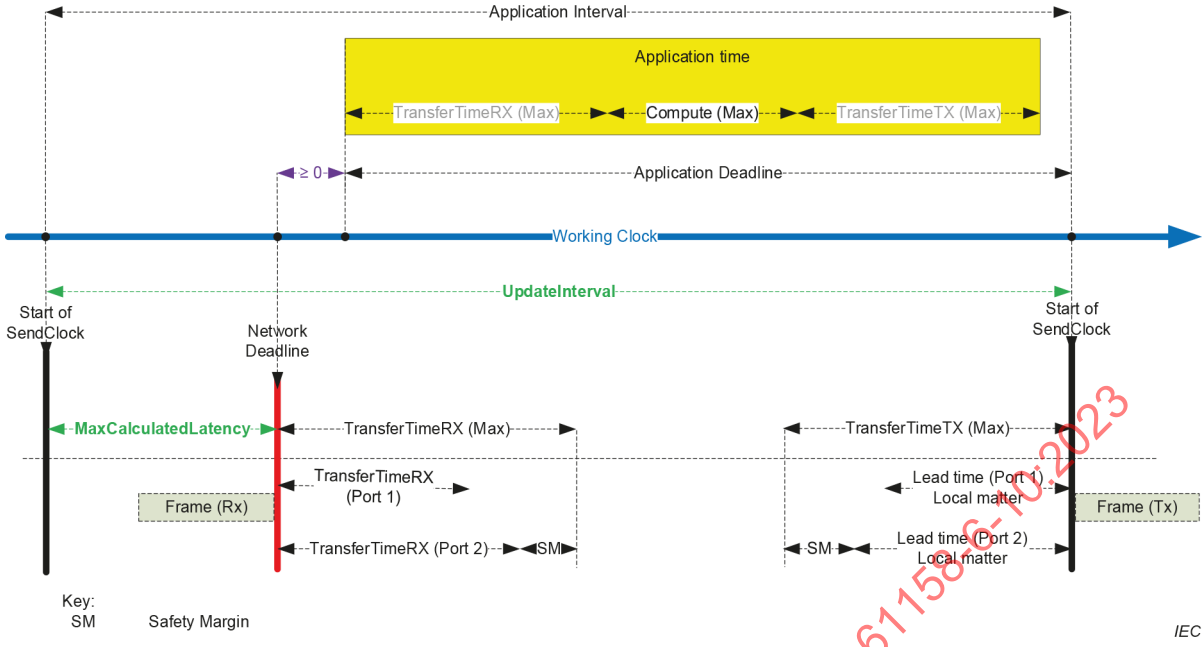


Figure 177 – Timing model with RR = 1

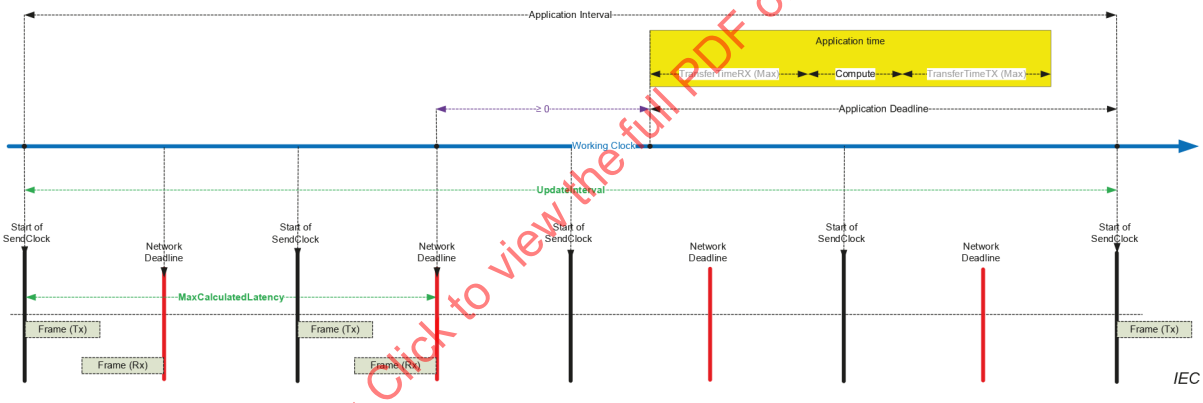


Figure 178 – Timing model with RR = 4

5.2.11.9.10 Coding of the field StreamType

This field shall be coded as data type Unsigned16 according to Table 779.

Table 779 – StreamType

Value (hexadecimal)	Meaning
0x00	Unicast stream
0x01	Multicast stream
Other	Reserved

5.2.11.10 Coding of the field CIMStreamCollectionUUID

This field shall be coded as data type UUID. All possible values are valid. The value shall be given to the stream collection with each write service.

5.2.11.11 Coding of the field CIMSyncTreeDataUUID

This field shall be coded as data type UUID. All possible values are valid. The value shall be given to the stream collection with each write service.

5.2.11.12 Coding of the field CIMExpectedNetworkAttributesUUID

This field shall be coded as data type UUID. All possible values are valid. The value shall be given to the stream collection with each write service.

5.2.12 Coding section related to CIM

5.2.12.1 Coding section related to Physical Device IR Data

5.2.12.1.1 Coding of the field RxPort

This field shall be coded as data type Unsigned8. This field shall be coded with the values according to Table 780.

Table 780 – RxPort

Value (hexadecimal)	Meaning
0x00	Local interface Identifies the bridge component port connected to the end station component port and thus, the interface of this device.
0x01	Port 1
0x02	Port 2
…	…
0xFF	Port 255

5.2.12.1.2 Coding of the field NumberOfTxPortGroups

This field shall be coded as data type Unsigned8 and shall be set according to Table 781. This field shall only count the successing TxPortGroupArray entries.

Table 781 – NumberOfTxPortGroups

Value (hexadecimal)	Meaning
0x01, 0x03, 0x05, 0x07, 0x09, 0x0A, 0x0C, 0x0F, 0x11, 0x13, 0x15, 0x17, 0x19, 0x1A, 0x1C, 0x1F, 0x21	Allowed values
Other	Reserved

5.2.12.1.3 Coding of the field TxPortGroupArray

The field TxPortGroupArray is an array of octets that shall contain at least one and at most 33 octets referred to as TxPortGroup octet. A TxPortGroup octet shall consist of at least one and at most 8 TxPort entries referred to as TxPortGroup entry 0 to TxPortGroup entry 7. Therefore, the number of TxPortGroup octets corresponds to the number of ports within a device and shall be calculated as follows:

$$N = M_{highest} \text{ DIV } 8 + 1 \quad (69)$$

where

N is the number of TxPortGroup octets or the number of array elements

$M_{highest}$ is the highest number of TxPorts within a device (maximum 255)

8 is the bit size of an octet

$$G = m \text{ DIV } 8 + 1 \quad (70)$$

where

G is the number of the TxPortGroup octet containing the TxPort

m is the number of the current TxPort with 1 ≤ m ≤ M

8 is the bit size of an octet

$$B = m \text{ MOD } 8 \quad (71)$$

where

B is the number of the bit in the TxPortGroup octet containing the TxPort

m is the number of the current TxPort with 1 ≤ m ≤ M

8 is the bit size of an octet

NOTE The term DIV stands for division without remainder. The term MOD stands for the remainder of the division.

The last TxPortGroup octet (octet N) does not in every case contain all 8 TxPort entries. If Mhighest MOD 8 ≠ 7 the octet N is not fully filled and the remaining bits shall be set to zero referred to as Padding Bits.

The TxPortGroup of the devices TxPorts shall be structured in ascending order without gaps. The coding of this field shall be according to 3.4.2.3 and the individual bits shall have the following meaning:

Bit 0: TxPortEntry_0
This bit shall be set with the values according to Table 782 if the TxPort with the number that meets m MOD 8 = 0 is present. A padding bit shall be used if no TxPort is present.

Table 782 – TxPortEntry

Value (hexadecimal)	Meaning
0x00	Transmission off
0x01	Transmission on

The TxPort of local injection is always placed in TxPortEntry_0 of the TxPortGroup octet number one.

Bit 1: TxPortEntry_1
This bit shall be set with the values according to Table 782 if the TxPort with the number that meets m MOD 8 = 1 is present. A padding bit shall be used if no TxPort is present.

Bit 2: TxPortEntry_2
This bit shall be set with the values according to Table 782 if the TxPort with the number that meets m MOD 8 = 2 is present. A padding bit shall be used if no TxPort is present.

Bit 3: TxPortEntry_3
This bit shall be set with the values according to Table 782 if the TxPort with the number that meets m MOD 8 = 3 is present. A padding bit shall be used if no TxPort is present.

Bit 4: TxPortEntry_4
This bit shall be set with the values according to Table 782 if the TxPort with the number that meets m MOD 8 = 4 is present. A padding bit shall be used if no TxPort is present.

Bit 5: TxPortEntry_5
This bit shall be set with the values according to Table 782 if the TxPort with the number that meets m MOD 8 = 5 is present. A padding bit shall be used if no TxPort is present.

Bit 6: TxPortEntry_6
This bit shall be set with the values according to Table 782 if the TxPort with the number that meets m MOD 8 = 6 is present. A padding bit shall be used if no TxPort is present.

Bit 7: TxPortEntry_7
This bit shall be set with the values according to Table 782 if the TxPort with the number that meets m MOD 8 = 7 is present. A padding bit shall be used if no TxPort is present.

5.2.12.1.4 Coding of the field FrameDetails

This field shall be coded according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0 – 1: FrameDetails.SyncFrame
This field shall be coded with the values according to Table 783 and Table 784.

Table 783 – FrameDetails.SyncFrame in conjunction with FrameDataProperties.ForwardingMode==“Absolute mode”

Value (hexadecimal)	Meaning	Description
0x00	No sync frame	Mandatory
0x01	Primary sync frame	Legacy
0x02	Secondary sync frame	Primary and secondary sync frame use the same FrameID. At one time only one frame shall be valid.
0x03	Reserved	—

Table 784 – FrameDetails.SyncFrame in conjunction with FrameDataProperties.ForwardingMode==“Relative mode”

Value (hexadecimal)	Meaning	Description
0x00	No sync frame	Mandatory
Other	Reserved	—

Bit 2 – 3: FrameDetails.MeaningFrameSendOffset
This field shall be coded with the values according to Table 785.

Table 785 – FrameDetails.MeaningFrameSendOffset

Value (hexadecimal)	Meaning	Usage
0x00	The content of the field FrameSendOffset specifies the point of time for transmitting a frame. For the receiver the content of the field specifies the point of time for receiving a frame.	Mandatory
other	Reserved	—

Bit 4 – 6: FrameDetails.Reserved
This field shall be set to zero.

Bit 7: FrameDetails.MediaRedundancyWatchDog
This field shall be set according to Table 786.

Table 786 – FrameDetails.MediaRedundancyWatchDog

Value (hexadecimal)	Meaning	Meaning
0x00	Disable	Do not monitor the quality of a MRPD connection
0x01	Enable	Monitor the quality of a MRPD connection and signal a change

5.2.12.1.5 Coding of the field FrameDataProperties

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0: FrameDataProperties.ForwardingMode
This field shall be set according to Table 787.

Table 787 – FrameDataProperties.ForwardingMode

Value (hexadecimal)	Meaning
0x00	Absolute mode The FrameSendOffset contains the send time of a forwarded frame
0x01	Relative mode The FrameSendOffset shall contain the send time of a forwarded frame or the “Best effort” value.

Bit 1 – 2: FrameDataProperties.FastForwardingMulticastMACAdd
This field shall be set according to Table 788.

Table 788 – FrameDataProperties.FastForwardingMulticastMACAdd

Value (hexadecimal)	Meaning
0x00	Legacy Use interface MAC destination unicast address
0x01	Use RT_CLASS_3 destination multicast address
0x02	Use FastForwardingMulticastMACAdd
0x03	Reserved

Bit 3 – 4: FrameDataProperties.FragmentationMode
This field shall be coded with the values according to Table 789.

Table 789 – FrameDataProperties.FragmentationMode

Value (hexadecimal)	Meaning	Usage
0x00	No fragmentation	Mandatory
0x01	Fragmentation enabled Maximum fragment size for static fragmentation: 128 octets	Optional
0x02	Fragmentation enabled Maximum fragment size for static fragmentation: 256 octets	Optional
0x03	Reserved	—

Bit 5 – 15: FrameDataProperties.Reserved_1
This field shall be set to zero.

Bit 16 – 31: FrameDataProperties.Reserved_2
This field shall be set according to 3.4.2.2.

5.2.12.1.6 Coding of the field MaxBridgeDelay

This field shall be coded as data type Unsigned32 according to Table 790, in nanoseconds. Figure 43 shows the meaning of MaxBridgeDelay.

NOTE The bridge or forwarding delay between port 0 and any other port, and the bridge or forwarding delay between any port (not port 0) can differ.

Table 790 – MaxBridgeDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x0000FFFF	Bridge delay used by engineering for RT_CLASS_3 calculation
0x00010000 – 0xFFFFFFFF	Reserved

5.2.12.1.7 Coding of the field NumberOfPorts

This field shall be coded as data type Unsigned32 with values according to Table 791.

Table 791 – NumberOfPorts

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x000000FF	Number of following port entries
0x00000100 – 0xFFFFFFFF	Reserved

5.2.12.1.8 Coding of the field MaxPortTxDelay

This field shall be coded as data type Unsigned32 according to Table 792, in nanoseconds. Figure 43 shows the meaning of MaxPortTxDelay.

Table 792 – MaxPortTxDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x0000FFFF	Port transmit delay used by engineering for RT_CLASS_3 calculation
0x00010000 – 0xFFFFFFFF	Reserved

5.2.12.1.9 Coding of the field MaxPortRxDelay

This field shall be coded as data type Unsigned32 according to Table 793, in nanoseconds. Figure 43 shows the meaning of MaxPortRxDelay.

Table 793 – MaxPortRxDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x0000FFFF	Port receive delay used by engineering for RT_CLASS_3 calculation
0x00010000 – 0xFFFFFFFF	Reserved

5.2.12.1.10 Coding of the field MaxLineRxDelay

This field shall be coded as data type Unsigned32 according to Table 794, in nanoseconds. Figure 43 shows the meaning of MaxLineRxDelay.

Table 794 – MaxLineRxDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown No delay, forward as soon as possible
0x00000001 – 0x3B9AC9FF	Line delay used by engineering for RT_CLASS_3 calculation
0x3B9ACA00 – 0xFFFFFFFF	Reserved

5.2.12.1.11 Coding of the field YellowTime

This field shall be coded as data type Unsigned32 according to Table 795, in nanoseconds. Figure 179 and Figure 185 show the meaning and Figure 180 the calculation principle of YellowTime.

The YELLOW period allows a bridge to optimize the forwarding of frames as shown in Figure 180.

If the YellowTime, calculated according to Formula (72) and (73), is less than the maximum Ethernet frame, the fragmentation shall be enabled.

NOTE The YellowTime is 125 µs if running without fragmentation.

$$\text{YellowTime} = (\text{InterFrameGAP} + \text{PreambleLength} + \text{StartFrameDelimiter} + \text{MAX(Selected Protocols)}) \times 80\ \text{ns} + \text{REDBeginSafetyMargin} \quad (72)$$

where

YellowTime	is the minimum time between two reserved periods
InterFrameGAP	is the minimum distance between two frames
PreambleLength	is the length of the Preamble
StartFrameDelimiter	is the time used to transmit the SFD
MAX(Selected Protocols)	is the maximum frame size of the protocol list
REDBeginSafetyMargin	is the time which covers internal delays of a switch

$$\text{MAX(Selected Protocols)} = \text{Maximum (PTCP, MRP, RT, IEEE Std 802.1AS)} \quad (73)$$

where

MAX(Selected Protocols)	is the maximum frame size of the protocol list
PTCP	is the maximum size of a frame used by the PTCP protocol. Fragmentable protocol elements are ignored
MRP	is the maximum size of a frame used by the MRP protocol
RT	is the maximum frame length of a RT_CLASS_1/_2 frame used in this specific environment
IEEE Std 802.1AS	is the maximum size of a frame used by the IEEE Std 802.1AS protocol. Fragmentable protocol elements are ignored

Table 795 – YellowTime

Value (hexadecimal)	Meaning	Usage
0x00 – 0x1ADF	Reserved	—
0x1AE0	Yellow time in nanoseconds Equal to 86 octets at 100 Mbit/s	Fragmentation is supported. In this case Formula (73) may return the value 86 octets.
0x1AE1 – 0x1E847	—	Fragmentation is supported. The frame length of RT is between 87 and 1 521 octets.
0x1E848	Default Yellow time in nanoseconds Equal to 1 562,5 octets at 100 Mbit/s	Fragmentation is not supported. In this case Formula (73) returns the value 1 522 octets.
0x0x1E849 – 0xFFFFFFFF	Reserved	—

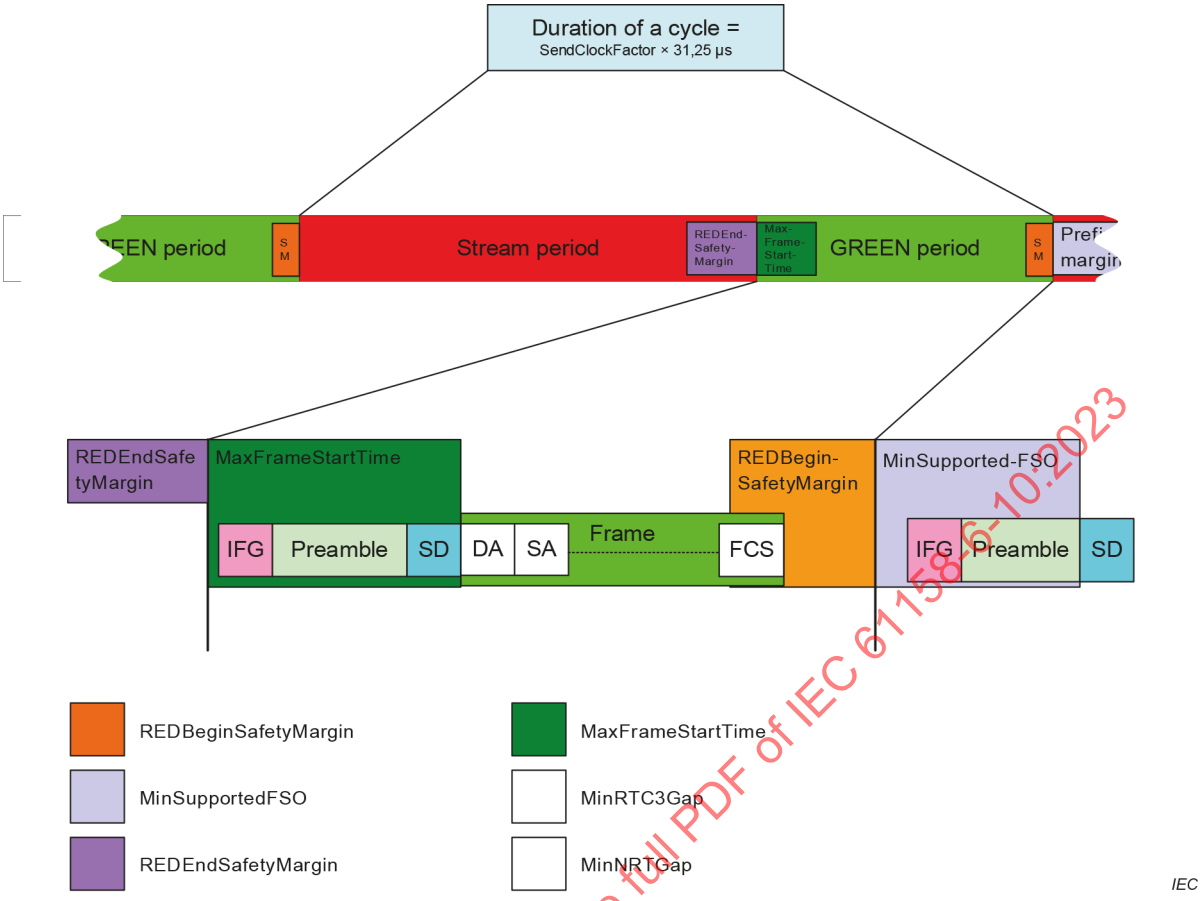


Figure 179 – Calculation principle for a cycle

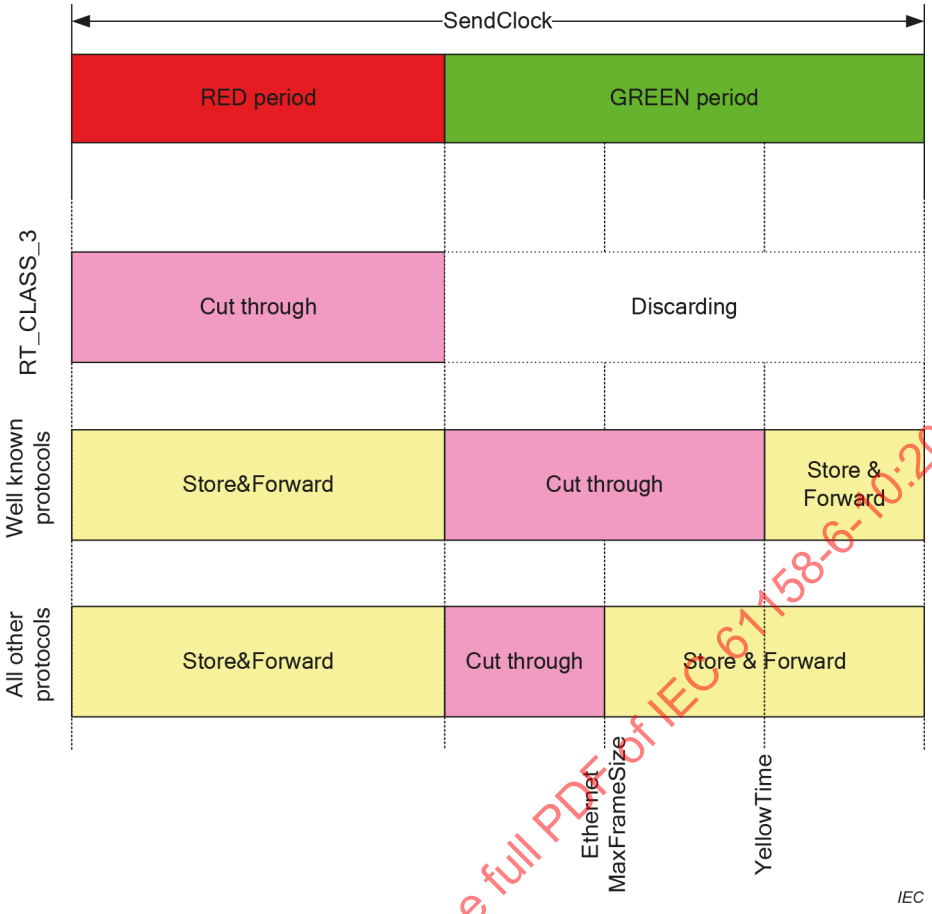


Figure 180 – Calculation principle for the minimum YellowTime

5.2.12.1.12 Coding of the field StartOfRedFrameID

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 796.

All RT_CLASS_3 frames, forwarded or local injected, shall be inside the range given by StartOfRedFrameID and EndOfRedFrameID.

Table 796 – StartOfRedFrameID

Value (hexadecimal)	Meaning
0x0000 – 0x00FF	Reserved
0x0100 – 0x0FFF	Mandatory 0x0100 is the default value
0x1000 – 0xFFFF	Reserved

5.2.12.1.13 Coding of the field EndOfRedFrameID

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 797 and Table 798.

Table 797 – EndOfRedFrameID

Value (hexadecimal)	Meaning
0x0000 – 0x00FF	Reserved
0x0100 – 0x0FFF	Mandatory 0x0FFF is the default value
0x1000 – 0xFFFF	Reserved

Table 798 – Dependencies of StartOfRedFrameID and EndOfRedFrameID

Value (hexadecimal)	Meaning
StartOfRedFrameID ≤ EndOfRedFrameID	Valid Shall cover the needed FrameID range
StartOfRedFrameID > EndOfRedFrameID	Not used

5.2.12.1.14 Coding of the field NumberOfAssignments

This field shall be coded as data type Unsigned32. This field shall be coded with the values according to Table 799.

Table 799 – NumberOfAssignments

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x00000004	Mandatory, if only phases with reserved periods (RED) exist, four entries shall be supported. Number of following assignment entries
0x00000005	Mandatory, if phases without reserved periods (RED) exist, five entries shall be supported. Number of following assignment entries
0x00000006 – 0x0000000010	Optional Number of following assignment entries
0x00000011 – 0xFFFFFFFF	Reserved

If there is no reserved period for a port, then the NumberOfAssignments shall be coded with 1 and RedOrangePeriodBegin, OrangePeriodBegin and GreenPeriodBegin shall be coded with zero. Furthermore, the NumberOfPhases shall be coded with 1 and AssignedValueForReservedBegin, AssignedValueForOrangeBegin and AssignedValueForReservedEnd shall be coded with zero.

5.2.12.1.15 Coding of the field NumberOfPhases

This field shall be coded as data type Unsigned32. This field shall be coded with the values according to Table 800.

Table 800 – NumberOfPhases

Value (hexadecimal)	Meaning
Other	Reserved
0x00000001	Mandatory One following phase assignment entry
0x00000002	Mandatory Two following phase assignment entries
0x00000004	Mandatory Four following phase assignment entries
0x00000008	Mandatory Eight following phase assignment entries
0x00000010	Mandatory Sixteen following phase assignment entries

NOTE If more than 16 phases exist due to a ReductionRatio greater sixteen, the engineering shall logically reduce the amount of begin and end values to fit again in this structure.

5.2.12.1.16 Coding of the field PhaseAssignment

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 3: AssignedValueForReservedBegin
This bit shall be set with the values according to Table 801.

Table 801 – AssignedValueForReservedBegin

Value (hexadecimal)	Meaning
0x00	RedOrangePeriodBegin of entry one of the XXBeginEndAssignment shall be used
0x01 – 0x0E	…
0x0F	RedOrangePeriodBegin of entry sixteen of the XXBeginEndAssignment shall be used

Bit 4 – 7: AssignedValueForOrangeBegin
This bit shall be set with the values according to Table 802.

Table 802 – AssignedValueForOrangeBegin

Value (hexadecimal)	Meaning
0x00	OrangePeriodBegin of entry one of the XXBeginEndAssignment shall be used
0x01 – 0x0E	…
0x0F	OrangePeriodBegin of entry sixteen of the XXBeginEndAssignment shall be used

Bit 8 – 11: AssignedValueForReservedEnd
This bit shall be set with the values according to Table 803.

Table 803 – AssignedValueForReservedEnd

Value (hexadecimal)	Meaning
0x00	GreenPeriodBegin of entry one of the XXBeginEndAssignment shall be used
0x01 – 0x0E	…
0x0F	GreenPeriodBegin of entry sixteen of the XXBeginEndAssignment shall be used

Bit 12 – 15: Reserved
This bit shall be set to zero.

5.2.12.1.17 Coding of the field RedOrangePeriodBegin

This field shall be coded as data type Unsigned32 with values according to Table 804 and Table 805. The time base shall be one nanosecond.

Table 804 – Values of RedOrangePeriodBegin

Value (hexadecimal)	Meaning	Use
0	Relative offset from the start of the related cycle	Mandatory
1 – 0x003D08FF	Relative offset from the start of the related cycle	Optional Not recommended
0x003D0900 – 0xFFFFFFFF	Reserved	—

Table 805 – Dependencies of RedOrangePeriodBegin, OrangePeriodBegin and GreenPeriodBegin

Value (hexadecimal)	Meaning	Usage
RedOrangePeriodBegin < OrangePeriodBegin < GreenPeriodBegin	Not used	—
RedOrangePeriodBegin = OrangePeriodBegin < GreenPeriodBegin	Not used	—
RedOrangePeriodBegin < OrangePeriodBegin = GreenPeriodBegin	Only a RED period is defined	Mandatory
RedOrangePeriodBegin = OrangePeriodBegin = GreenPeriodBegin = 0	No reserved period is defined	Mandatory
RedOrangePeriodBegin > OrangePeriodBegin	Not used	—
RedOrangePeriodBegin > GreenPeriodBegin	Not used	—
OrangePeriodBegin > GreenPeriodBegin	Not used	—

5.2.12.1.18 Coding of the field OrangePeriodBegin

This field shall be coded as data type Unsigned32 with values according to Table 805 and Table 806. The time base shall be one nanosecond.

Table 806 – Values of OrangePeriodBegin

Value (hexadecimal)	Meaning	Use
GreenPeriodBegin	OrangePeriodBegin shall be equal to GreenPeriodBegin	—
other	Reserved	—

5.2.12.1.19 Coding of the field GreenPeriodBegin

This field shall be coded as data type Unsigned32 with values according to Table 805 and Table 807. The time base shall be one nanosecond.

Table 807 – Values of GreenPeriodBegin

Value (hexadecimal)	Meaning	Use
0 – 0x0007A120	Relative offset from the start of the related cycle	Mandatory
0x0007A121 – 0x003D08FF	Relative offset from the start of the related cycle	Optional
0x003D0900 – 0xFFFFFFFF	Reserved	—

5.2.12.2 Coding section related to Physical Device Interface Data

5.2.12.2.1 Coding of the field MultipleInterfaceMode

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0: MultipleInterfaceMode.NameOfDevice
This field shall be set according to Table 808.

Table 808 – MultipleInterfaceMode.NameOfDevice

Value (hexadecimal)	Meaning
0x00	LLDPChassis identifies an IEEE Std 802 station if only one end station component is implemented in this station. The field LLDPChassis contains the NameOfStation and the field PortID of LLDP contains the name of the port.
0x01	LLDPChassis identifies an IEEE Std 802 station. The number of implemented end station and/or bridge components is not limited. The field LLDPChassis contains the NameOfDevice and is defined by local means. The field PortID of LLDP contains the name of the port and the NameOfStation.

Bit 1 – 15: MultipleInterfaceMode.Reserved_1
This field shall be set to zero.

Bit 16 – 31: MultipleInterfaceMode.Reserved_2
This field shall be set according to 3.4.2.2.

5.2.12.3 Coding section related to Physical Device Port Data

5.2.12.3.1 Coding of the field OwnPortName

This field shall be coded as data type OctetString[] according to 4.3.1.4.17.

5.2.12.3.2 Coding of the field LengthOwnPortName

This field shall be coded as data type Unsigned8 according to 4.3.1.4.17.

5.2.12.3.3 Coding of the field NumberOfPeers

This field shall be coded as data type Unsigned8 according to Table 809 and Table 810.

Table 809 – NumberOfPeers in conjunction with PDPortDataCheck or CIMNetConfExpectedNetworkAttributes

Value (hexadecimal)	Meaning
0x01	One neighbor shall be checked
Other	Reserved

Table 810 – NumberOfPeers in conjunction with PDPortDataReal or PDPortDataRealExtended

Value (hexadecimal)	Meaning
0x00	No neighbor exists or no LLDP frame received
0x01	One neighbor exists
0x02 – 0x04	Recommended Multiple neighbors
0x05 – 0xFF	Optional Multiple neighbors

5.2.12.3.4 Coding of the field LengthPeerPortName

This field shall be coded as data type Unsigned8.

5.2.12.3.5 Coding of the field PeerPortName

5.2.12.3.5.1 Usage in conjunction with PDPortDataCheck.CheckPeers

This field shall be coded according to 4.3.1.4.17.

5.2.12.3.5.2 Usage in conjunction with PDPortDataReal or PDPortDataRealExtended

This field shall be coded according to 4.3.1.4.17.

If the received LLDP_PortID satisfies the coding according to Table 376, use its NameOfPort part, otherwise the LengthPeerPortName should be zero.

Due to legacy implementations, coding differing from this document may be delivered in the fields PeerPortName and PeerStationName.

5.2.12.3.6 Coding of the field LengthPeerStationName

This field shall be coded as data type Unsigned8.

5.2.12.3.7 Coding of the field PeerStationName

5.2.12.3.7.1 General

This field shall be coded as data type OctetString[] with 1 to 255 octets.

5.2.12.3.7.2 Usage in conjunction with PDPortDataCheck.CheckPeers

This field shall be coded according to 4.3.1.4.16.

5.2.12.3.7.3 Usage in conjunction with PDPortDataReal or PDPortDataRealExtended

This field shall be coded according to 4.3.1.4.16.

If the received LLDP_PortID satisfies the coding according to Table 376, use the NameOfStation part from either LLDP_PortID or LLDP_ChassisID, otherwise the LengthPeerStationName should be zero.

Due to legacy implementations, coding differing from this document may be delivered in the fields PeerPortName and PeerStationName.

5.2.12.3.8 Coding of the field LengthOwnStationName

This field shall be coded as data type Unsigned8 according to 4.3.1.4.16.

5.2.12.3.9 Coding of the field OwnStationName

This field shall be coded as data type OctetString with 0 to 240 octets according to 4.3.1.4.16.

5.2.12.3.10 Coding of the field LineDelay

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 30: LineDelay.Value
This field shall be set according to Table 811, Table 812, Figure 42 and Formula (22).

Table 811 – LineDelay.Value with LineDelay.FormatIndicator == 0

Value (hexadecimal)	Meaning
0x00000000	Line delay and cable delay unknown
0x00000001 – 0x7FFFFFFF	Line delay in nanoseconds

Table 812 – LineDelay.Value with LineDelay.FormatIndicator == 1

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x7FFFFFFF	Cable delay in nanoseconds

Bit 31: LineDelay.FormatIndicator
This field shall be set according to Table 813.

Table 813 – LineDelay.FormatIndicator

Value (hexadecimal)	Meaning	Usage
0x00	LineDelay.Value is coded as line delay	Default, if line delay or cable delay is unknown Shall be used in conjunction with PDPortDataCheck, and if line delay or cable delay is unknown in conjunction with PDPortDataReal or PDPortDataRealExtended
0x01	LineDelay.Value is coded as cable delay	Default, if cable delay is known, together with IEEE Std 802.1AS Shall be used in conjunction with PDPortDataReal or PDPortDataRealExtended

5.2.12.3.11 Coding of the field PeerMACAddress

This field shall be coded as data type OctetString[6]. The value of the field PeerMACAddress shall be according to the 48-bit universal LAN MAC addresses of IEEE Std 802.

NOTE Octet 1 contains the Individual/Group Address Bit (lsb).

5.2.12.3.12 Coding of the field MAUType

This field shall be coded as data type Unsigned16 with the values according to IETF RFC 4836, Table 814, Table 815, and Table 816.

Table 814 – MAUType

Value (hexadecimal)	Meaning	Non-time-aware system Usage	Time-aware system Usage
0x0000	Reserved for MediaType “Copper cable” and “Fiber optic cable”	Operational Default for LinkState.Link != “UP”	Operational Default for LinkState.Link != “UP”
	Used for MediaType “Radio communication”	Operational Default for radio communication	Operational Default for radio communication
0x0005	10BaseT	Operational	Operational
0x000A	10BaseTXHD	Operational	Operational
0x000B	10BaseTXFD	Operational Administrative Observable	Operational Administrative Observable
0x000C	10BaseFLHD	Operational	Operational
0x000D	10BaseFLFD	Operational	Operational
0x000F	100BaseTXHD	Operational	Operational
0x0010	Default (MediaType Copper) 100BaseTXFD	Operational Administrative Observable	Operational Administrative Observable
0x0011	100BaseFXHD	Operational	Operational

Value (hexadecimal)	Meaning	Non-time-aware system Usage	Time-aware system Usage
0x0012	Default (MediaType Fiber optic) 100BaseFXFD	Operational Administrative Observable	Operational Administrative Observable
0x0015	1000BaseXHD	Operational	Operational
0x0016	1000BaseXFD	Operational Administrative Observable	Operational Administrative Observable
0x0017	1000BaseLXHD	Operational	Operational
0x0018	1000BaseLXFD	Operational Administrative Observable	Operational Administrative Observable
0x0019	1000BaseSXHD	Operational	Operational
0x001A	1000BaseSXFD	Operational Administrative Observable	Operational Administrative Observable
0x001D	1000BaseTHD	Operational	Operational
0x001E	1000BaseTFD	Operational Administrative Observable	Operational Administrative Observable
0x001F	10GbaseX	Operational Administrative Observable	Operational Administrative Observable
0x0020	10GbaseLX4	Operational Administrative Observable	Operational Administrative Observable
0x0021	10GbaseR	Operational Administrative Observable	Operational Administrative Observable
0x0022	10GbaseER	Operational Administrative Observable	Operational Administrative Observable
0x0023	10GbaseLR	Operational Administrative Observable	Operational Administrative Observable
0x0024	10GbaseSR	Operational Administrative Observable	Operational Administrative Observable
0x0025	10GbaseW	Operational Administrative Observable	Operational Administrative Observable
0x0026	10GbaseEW	Operational Administrative Observable	Operational Administrative Observable

Value (hexadecimal)	Meaning	Non-time-aware system Usage	Time-aware system Usage
0x0027	10GbaseLW	Operational Administrative Observable	Operational Administrative Observable
0x0028	10GbaseSW	Operational Administrative Observable	Operational Administrative Observable
0x0029	10GbaseCX4	Operational Administrative Observable	Operational Administrative Observable
0x002A	2BaseTL	Operational Administrative Observable	Operational Administrative Observable
0x002B	10PassTS	Operational Administrative Observable	Operational Administrative Observable
0x002C	100BaseBX10D	Operational Administrative Observable	Operational Administrative Observable
0x002D	100BaseBX10U	Operational Administrative Observable	Operational Administrative Observable
0x002E	100BaseLX10	Operational Administrative Observable	Operational Administrative Observable
0x002F	1000BaseBX10D	Operational Administrative Observable	Operational Administrative Observable
0x0030	1000BaseBX10U	Operational Administrative Observable	Operational Administrative Observable
0x0031	1000BaseLX10	Operational Administrative Observable	Operational Administrative Observable
0x0032	1000BasePX10D	Operational Administrative Observable	Operational Administrative Observable
0x0033	1000BasePX10U	Operational Administrative Observable	Operational Administrative Observable
0x0034	1000BasePX20D	Operational Administrative Observable	Operational Administrative Observable

Value (hexadecimal)	Meaning	Non-time-aware system Usage	Time-aware system Usage
0x0035	1000BasePX20U	Operational Administrative Observable	Operational Administrative Observable
0x0036	10GbaseT or 100BasePXFD	Operational Administrative Observable	Operational Administrative Observable
0x0037	10GbaseLRM	Operational Administrative Observable	Operational Administrative Observable
0x0038	1000BaseKX	Operational Administrative Observable	Operational Administrative Observable
0x0039	1000BaseKX4	Operational Administrative Observable	Operational Administrative Observable
0x003A	1000BaseKR	Operational Administrative Observable	Operational Administrative Observable
0x003B	10G1GbasePRXD1	Operational Administrative Observable	Operational Administrative Observable
0x003C	10G1GbasePRXD2	Operational Administrative Observable	Operational Administrative Observable
0x003D	10G1GbasePRXD3	Operational Administrative Observable	Operational Administrative Observable
0x003E	10G1GbasePRXU1	Operational Administrative Observable	Operational Administrative Observable
0x003F	10G1GbasePRXU2	Operational Administrative Observable	Operational Administrative Observable
0x0040	10G1GbasePRXU3	Operational Administrative Observable	Operational Administrative Observable
0x0041	10GbasePRD1	Operational Administrative Observable	Operational Administrative Observable
0x0042	10GbasePRD2	Operational Administrative Observable	Operational Administrative Observable

Value (hexadecimal)	Meaning	Non-time-aware system Usage	Time-aware system Usage
0x0043	10GbasePRD3	Operational Administrative Observable	Operational Administrative Observable
0x0044	10GbasePRU1	Operational Administrative Observable	Operational Administrative Observable
0x0045	10GbasePRU3	Operational Administrative Observable	Operational Administrative Observable
0x0046	40GbaseKR4	Operational	Operational
0x0047	40GbaseCR4	Operational	Operational
0x0048	40GbaseSR4	Operational	Operational
0x0049	40GbaseFR	Operational	Operational
0x004A	40GbaseLR4	Operational	Operational
0x004B	100GbaseCR10	Operational	Operational
0x004C	100GbaseSR10	Operational	Operational
0x004D	100GbaseLR4	Operational	Operational
0x004E	100GbaseER4	Operational	Operational
0x004F	1000BaseT1	Operational Administrative Observable	Operational Administrative Observable
0x0050	1000BasePX30D	Operational Administrative Observable	Operational Administrative Observable
0x0051	1000BasePX30U	Operational Administrative Observable	Operational Administrative Observable
0x0052	1000BasePX40D	Operational Administrative Observable	Operational Administrative Observable
0x0053	1000BasePX40U	Operational Administrative Observable	Operational Administrative Observable
0x0054	10G1GbasePRXD4	Operational Administrative Observable	Operational Administrative Observable
0x0055	10G1GbasePRXU4	Operational Administrative Observable	Operational Administrative Observable
0x0056	10GbasePRD4	Operational Administrative Observable	Operational Administrative Observable

Value (hexadecimal)	Meaning	Non-time-aware system Usage	Time-aware system Usage
0x0057	10GbasePRU4	Operational Administrative Observable	Operational Administrative Observable
0x0058	25GbaseCR	Operational	Operational
0x0059	25GbaseCRS	Operational	Operational
0x005A	25GbaseKR	Operational	Operational
0x005B	25GbaseKRS	Operational	Operational
0x005C	25GbaseR	Operational	Operational
0x005D	25GbaseSR	Operational	Operational
0x005E	25GbaseT	Operational	Operational
0x005F	40GbaseER4	Operational	Operational
0x0060	40GbaseR	Operational	Operational
0x0061	40GbaseT	Operational	Operational
0x0062	100GbaseCR4	Operational	Operational
0x0063	100GbaseKR4	Operational	Operational
0x0064	100GbaseKP4	Operational	Operational
0x0065	100GbaseR	Operational	Operational
0x0066	100GbaseSR4	Operational	Operational
0x0067	2.5Gbase-T	Operational Administrative Observable	Operational Administrative Observable
0x0068	5Gbase-T	Operational Administrative Observable	Operational Administrative Observable
0x0069	100Base-T1	Operational Administrative Observable	Operational Administrative Observable
0x006A	1000Base-RHA	Operational Administrative Observable	Operational Administrative Observable
0x006B	1000Base-RHB	Operational Administrative Observable	Operational Administrative Observable
0x006C	1000Base-RHC	Operational Administrative Observable	Operational Administrative Observable
0x006D	2.5Gbase-KX	Operational Administrative Observable	Operational Administrative Observable
0x006E	2.5Gbase-X	Operational Administrative Observable	Operational Administrative Observable

Value (hexadecimal)	Meaning	Non-time-aware system Usage	Time-aware system Usage
0x006F	5Gbase-KR	Operational Administrative Observable	Operational Administrative Observable
0x0070	5Gbase-R	Operational Administrative Observable	Operational Administrative Observable
0x0071	10Gpass-XR	Operational Administrative Observable	Operational Administrative Observable
0x0072	25Gbase-LR	Operational	Operational
0x0073	25Gbase-ER	Operational	Operational
0x0074	50Gbase-R	Operational	Operational
0x0075	50Gbase-CR	Operational	Operational
0x0076	50Gbase-KR	Operational	Operational
0x0077	50Gbase-SR	Operational	Operational
0x0078	50Gbase-FR	Operational	Operational
0x0079	50Gbase-LR	Operational	Operational
0x007A	50Gbase-ER	Operational	Operational
0x007B	100Gbase-CR2	Operational	Operational
0x007C	100Gbase-KR2	Operational	Operational
0x007D	100Gbase-SR2	Operational	Operational
0x007E	100Gbase-DR	Operational	Operational
0x007F	200Gbase-R	Operational	Operational
0x0080	200Gbase-DR4	Operational	Operational
0x0081	200Gbase-FR4	Operational	Operational
0x0082	200Gbase-LR4	Operational	Operational
0x0083	200Gbase-CR4	Operational	Operational
0x0084	200Gbase-KR4	Operational	Operational
0x0085	200Gbase-SR4	Operational	Operational
0x0086	200Gbase-ER4	Operational	Operational
0x0087	400Gbase-R	Operational	Operational
0x0088	400Gbase-SR16	Operational	Operational
0x0089	400Gbase-DR4	Operational	Operational
0x008A	400Gbase-FR8	Operational	Operational
0x008B	400Gbase-LR8	Operational	Operational
0x008C	400Gbase-ER8	Operational	Operational
0x008D	10Base-T1L	Operational Administrative Observable	Operational Administrative Observable
0x008E	10Base-T1SHD	Operational	Operational
0x008F	10Base-T1SMD	Operational	Operational

Value (hexadecimal)	Meaning	Non-time-aware system Usage	Time-aware system Usage
0x0090	10Base-T1SFD	Operational Administrative Observable	Operational Administrative Observable
Usable well-known numbers	Speed of 10 Mbit/s (and more) and full duplex mode	Operational Administrative Observable	Operational Administrative Observable
Unusable well-known numbers	E.g. half duplex mode	Operational	Operational
other	Reserved	—	—

For IEEE Std 802.3 MAU types, the usage of MAUTypeExtension should be avoided. Only MAU types defined in this document shall use MAUTypeExtension.

Due to legacy reasons, “100BasePXFD” and “100BaseFXFD, POF” (see Table 815) are two codings for the same MAU type.

Table 815 – MAUType with MAUTypeExtension

Value (hexadecimal)	Meaning	Usage
0x0012 [a], 0x0100 [b]	100BaseFXFD [a], POF [b]	PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x008D [a], 0x0200 [b]	10BaseT1L [a], APL [b]	PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
[a] See Table 814. [b] See Table 817.		

Table 816 – Valid combinations between MAUType and LinkState

LinkState.Port	LinkState.Link	MAUType	Usage
Unknown Disabled / Discarding Blocking Listening Learning Forwarding Broken	Up (ready to pass packets)	10BaseT	PDPortDataReal
		10BaseTXHD	
		10BaseTXFD	
		10BaseFLHD	
		10BaseFLFD	
		100BaseTXHD	
		100BaseTXFD (Default)	
		100BaseFXHD	
		100BaseFXFD	
		1000BaseXHD	
		1000BaseXFD	
		1000BaseLXHD	
		1000BaseLXFD	
		1000BaseSXHD	
		1000BaseSXFD	

LinkState.Port	LinkState.Link	MAUType	Usage
		1000BaseTHD	
		1000BaseTFD	
		10GbaseFX	
		100BaseLX10	
		100BasePXFD	
		Usable well known numbers	
		Unusable well known numbers	
	Down	Reserved	
	Testing (in some test mode)		
	Unknown (status cannot be determined)		
Reserved	Reserved		

5.2.12.3.13 Coding of the field MAUTypeExtension

This field, defined as an administrative number, shall be coded as data type Unsigned16 with the values according to Table 817.

This document defines, in addition to the IEEE Std 802.3, additional MAU types. These MAUTypeExtensions are intended to be derivates of the MAU types.

Table 817 – MAUTypeExtensions and its corresponding MAUTypes

Value (hexadecimal)	Meaning	Attached MAUType	Comments
0x0000	No extension	Table 814	The MAUType from Table 814 applies.
0x0001 – 0x00FF	Reserved	—	—
0x0100	POF [a]	100BaseFXFD	Polymeric optical fiber
0x0101 – 0x01FF	Reserved for extensions	—	Central administrative number to unambiguously distinguish between MAUTypeExtensions.
0x0200	APL [b] (Advanced Physical Layer)	10BaseT1L	Intrinsic safety extended physical layer
0x0201 – 0xFEFF	Reserved for extensions	—	Central administrative number to unambiguously distinguish between MAUTypeExtensions.
0xFF00 – 0xFFFF	Reserved	—	—

[a] See IEC 61158-2.

[b] See IEC TS 60079-47.

5.2.12.3.14 Coding of the field AdjustProperties

This field shall be coded as data type Unsigned16 with the value zero.

5.2.12.3.15 Coding of the field CheckSyncMode

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0: CheckSyncMode.CableDelay
This field shall be set according to Table 818.

Table 818 – CheckSyncMode.CableDelay

Value (hexadecimal)	Meaning	Usage
0x00	OFF	No check
0x01	ON	Check cable delay difference between local and remote measured cable delay versus 50 ns.

Bit 1: CheckSyncMode.SyncMaster
This field shall be set according to Table 819.

Table 819 – CheckSyncMode.SyncMaster

Value (hexadecimal)	Meaning	Usage
0x00	OFF	No check
0x01	ON	Check PTCP_MasterSourceAddress between local and remote using LLDP_PNIO_PTCPSTATUS.

Bit 2 – 15: CheckSyncMode.Reserved
This field shall be set according to 3.4.2.2.

5.2.12.3.16 Coding of the field MAUTypeMode

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0: MAUTypeMode.Check
This field shall be set according to Table 820.

Table 820 – MAUTypeMode.Check

Value (hexadecimal)	Meaning	Usage
0x00	OFF	No check
0x01	ON	Check MAU type difference between local and remote detected value including MAUTypeExtension.

Bit 1 – 15: MAUTypeMode.Reserved
This field shall be set according to 3.4.2.2.

5.2.12.3.17 Coding of the field DomainBoundaryIngress

The coding of this field shall be according to 3.4.2.3.7 and the individual bits shall be coded with the values according to Table 821.

Table 821 – DomainBoundaryIngress

Bit	Value	Meaning
0	1	Block an incoming frame with the multicast MAC address 01-0E-CF-00-04-20, 01-0E-CF-00-04-40 and 01-0E-CF-00-04-80
	0	Do not block an incoming frame with the multicast MAC address 01-0E-CF-00-04-20, 01-0E-CF-00-04-40 and 01-0E-CF-00-04-80
1..31	Reserved	

5.2.12.3.18 Coding of the field DomainBoundaryEgress

The coding of this field shall be according to 3.4.2.3.7 and the individual bits shall be coded with the values according to Table 822.

Table 822 – DomainBoundaryEgress

Bit	Value	Meaning
0	1	Block an outgoing frame with the multicast MAC address 01-0E-CF-00-04-20, 01-0E-CF-00-04-40 and 01-0E-CF-00-04-80
	0	Do not block an outgoing frame with the multicast MAC address 01-0E-CF-00-04-20, 01-0E-CF-00-04-40 and 01-0E-CF-00-04-80
1..31	Reserved	

For the PTCP-AnnouncePDU Table 821 and Table 822 shall be combined according to Table 823 to decode the domain boundary.

Table 823 – DomainBoundaryAnnounce

—	Domain-Boundary-Ingress	Domain-Boundary-Egress	Meaning
Bit	Value	Value	—
0	1	1	Discard frames with the multicast MAC address 01-0E-CF-00-04-00
	0	1	Do not block a frame with the multicast MAC address 01-0E-CF-00-04-00
	1	0	
	0	0	
1..31	Reserved		

5.2.12.3.19 Coding of the field MulticastBoundary

This field shall be coded as data type Unsigned32. The individual bits shall be coded with the values according to Table 824.

This boundary shall be applied to the first 32 RT_CLASS_2 multicast addresses.

Table 824 – MulticastBoundary

Bit	Value	Meaning
0	1	Block the multicast MAC address 01-0E-CF-00-02-00
	0	Do not block the multicast MAC address 01-0E-CF-00-02-00
…	1	Block the multicast MAC address 01-0E-CF-00-02-xx
	0	Do not block the multicast MAC address 01-0E-CF-00-02-xx
31	1	Block the multicast MAC address 01-0E-CF-00-02-1F
	0	Do not block the multicast MAC address 01-0E-CF-00-02-1F

5.2.12.3.20 Coding of the field PeerToPeerBoundary

This field shall be coded as data type Unsigned32. The individual bits shall be coded with the values according to Table 825.

Table 825 – PeerToPeerBoundary

Bit	Value	Meaning
0	1	The LLDP agent shall not send LLDP frames (egress filter).
	0	The LLDP agent shall send LLDP frames for this port.
1	1	The PTCP ASE shall not send PTCP_DELAY request frames (egress filter).
	0	The PTCP ASE shall send PTCP_DELAY request frames for this port.
2	1	The Time ASE shall not send PATH_DELAY request frames (egress filter).
	0	The Time ASE shall send PATH_DELAY request frames for this port.
…	0	Reserved
31	0	Reserved

5.2.12.3.21 Coding of the field DCPBoundary

This field shall be coded as data type Unsigned32. The individual bits shall be coded with the values according to Table 826.

Table 826 – DCPBoundary

Bit	Value	Meaning
0	1	Block an outgoing DCP_Identify frame (egress filter) with the multicast MAC address 01-0E-CF-00-00-00
	0	Do not block the multicast MAC address 01-0E-CF-00-00-00
1	1	Block an outgoing DCP_Hello frame (egress filter) with the multicast MAC address 01-0E-CF-00-00-01
	0	Do not block the multicast MAC address 01-0E-CF-00-00-01
…	0	Reserved
31	0	Reserved

5.2.12.3.22 Coding of the field PreambleLength

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0: PreambleLength.Length
This field shall be set according to Table 827.

Table 827 – PreambleLength.Length

Value (hexadecimal)	Meaning	Usage
0x00	Seven octets Preamble shall be used	Default For the Ethernet frames the PHY shall use seven octets preamble before the start delimiter (SD)
0x01	One octet Preamble shall be used	For the Ethernet frames the PHY shall use one octet preamble before the start delimiter (SD)

Bit 1 – 15: PreambleLength.Reserved
This field shall be set according to 3.4.2.2.

5.2.12.3.23 Coding of the field LinkState

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: LinkState.Link
This field shall be set according to Table 828.

Table 828 – LinkState.Link

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
0x01	Up (ready to pass packets)	PDPortDataReal, CheckLinkState
0x02	Down	PDPortDataReal, AdjustLinkState
0x03	Testing (in some test mode)	PDPortDataReal
0x04	Unknown (status cannot be determined)	PDPortDataReal
0x05	Dormant	PDPortDataReal
0x06	NotPresent E.g. used if an SFP is not plugged	PDPortDataReal
0x07	LowerLayerDown	PDPortDataReal
0x08 – 0xFF	Reserved	—

Bit 8 – 15: LinkState.Port
This field shall be set according to Table 829. Its value shall be controlled by MRP, RSTP or any other supported loop prevention / media redundancy protocol, managing the active topology, used together with this document.

Table 829 – LinkState.Port

Value (hexadecimal)	Meaning	Usage
0x00	Unknown	Default Mandatory PDPortDataReal
0x01	Disabled / Discarding	Optional PDPortDataReal
0x02	Blocking	Optional PDPortDataReal
0x03	Listening	Optional PDPortDataReal
0x04	Learning	Optional PDPortDataReal
0x05	Forwarding	Optional PDPortDataReal
0x06	Broken	Optional PDPortDataReal
0x07 – 0xFF	Reserved	—

5.2.12.3.24 Coding of the field MediaType

This field shall be coded as data type Unsigned32 with the values according to Table 830.

Table 830 – MediaType

Value (hexadecimal)	Meaning	Usage
0x00	Unknown	PDPortDataReal
0x01	Copper cable	PDPortDataReal
0x02	Fiber optic cable	PDPortDataReal
0x03	Radio communication	PDPortDataReal
0x04 – 0xFFFFFFFF	Reserved	—

5.2.12.4 Coding section related to Physical Network Configuration Data

5.2.12.4.1 Coding of the field NMEDomainVIDConfig

The coding of this field shall be according to 3.4.2.3.7 and the individual bits shall have the following meaning:

Bit 0 – 11: NMEDomainVIDConfig.StreamHighVID
The bits shall be coded according to IEEE Std 802.1Q with the values according to Table 831.

Table 831 – NMEDomainVIDConfig.StreamHighVID

Value (decimal)	Meaning
0	Reserved According to IEEE Std 802.1Q definition, no VLAN assigned
101	Default, according to Table 415.
Others	IEEE Std 802.1Q definition applies.

Bit 12 – 23: NMEDomainVIDConfig.StreamHighRedVID
The bits shall be coded according to IEEE Std 802.1Q with the values according to Table 832.

Table 832 – NMEDomainVIDConfig.StreamHighRedVID

Value (decimal)	Meaning
0	Reserved According to IEEE Std 802.1Q definition, no VLAN assigned
102	Default, according to Table 415.
Others	IEEE Std 802.1Q definition applies.

Bit 24 – 35: NMEDomainVIDConfig.StreamLowVID
The bits shall be coded according to IEEE Std 802.1Q with the values according to Table 833.

Table 833 – NMEDomainVIDConfig.StreamLowVID

Value (decimal)	Meaning
0	Reserved According to IEEE Std 802.1Q definition, no VLAN assigned
103	Default, according to Table 415.
Others	IEEE Std 802.1Q definition applies.

Bit 36 – 47: NMEDomainVIDConfig.StreamLowRedVID
The bits shall be coded according to IEEE Std 802.1Q with the values according to Table 834.

Table 834 – NMEDomainVIDConfig.StreamLowRedVID

Value (decimal)	Meaning
0	Reserved According to IEEE Std 802.1Q definition, no VLAN assigned
104	Default, according to Table 415.
Others	IEEE Std 802.1Q definition applies.

Bit 48 – 59: NMEDomainVIDConfig.NonStreamVID
The bits shall be coded according to IEEE Std 802.1Q with the values according to Table 835.

Table 835 – NMEDomainVIDConfig.NonStreamVID

Value (decimal)	Meaning
0	Reserved According to IEEE Std 802.1Q definition, no VLAN assigned
100	Default VID for VLAN A, according to Table 415.
Others	IEEE Std 802.1Q definition applies.

Bit 60 – 71: NMEDomainVIDConfig.NonStreamVIDB
The bits shall be coded according to IEEE Std 802.1Q with the values according to Table 836.

Table 836 – NMEDomainVIDConfig.NonStreamVIDB

Value (decimal)	Meaning
0	Disabled, no passthrough VLAN assigned.
105	Default VID for VLAN B, according to Table 415.
Others	IEEE Std 802.1Q definition applies.

Bit 72 – 83: NMEDomainVIDConfig.NonStreamVIDC
The bits shall be coded according to IEEE Std 802.1Q with the values according to Table 837.

Table 837 – NMEDomainVIDConfig.NonStreamVIDC

Value (decimal)	Meaning
0	Disabled, no passthrough VLAN assigned.
106	Default VID for VLAN C, according to Table 415.
Others	IEEE Std 802.1Q definition applies.

Bit 84 – 95: NMEDomainVIDConfig.NonStreamVIDD
The bits shall be coded according to IEEE Std 802.1Q with the values according to Table 838.

Table 838 – NMEDomainVIDConfig.NonStreamVIDD

Value (decimal)	Meaning
0	Disabled, no passthrough VLAN assigned.
107	Default VID for VLAN D, according to Table 415.
Others	IEEE Std 802.1Q definition applies.

Bit 96 – 127: NMEDomainVIDConfig.Reserved
This field shall be set to zero.

5.2.12.4.2 Coding of the field NMEDomainQueueConfig

The coding of this field shall be according to 3.4.2.3.6 and the individual bits shall have the following meaning:

Bit 0 – 3: NMEDomainQueueConfig.QueueID
The bits shall be coded with the values according to Table 839.

Table 839 – NMEDomainQueueConfig.QueueID

Value (hexadecimal)	Meaning
0x00 – 0x07	Identifier of the queue
0x08 – 0x0F	Reserved

Bit 4 – 6: NMEDomainQueueConfig.TciPcp
The bits shall be coded according to IEEE Std 802.1Q with the values according to Table 840.

Table 840 – NMEDomainQueueConfig.TciPcp

Value (hexadecimal)	Meaning
0x00 – 0x07	TCI.PCP value as defined by the IEEE Std 802.1Q mapped to this queue.

Bit 7 – 13: NMEDomainQueueConfig.Shaper
The bits shall be coded with the values according to Table 841.

Table 841 – NMEDomainQueueConfig.Shaper

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Strict priority (SP)
0x02	Enhancements for Transmission Selection (ETS)
other	Reserved

Bit 14 – 15: NMEDomainQueueConfig.PreemptionMode
The bits shall be coded with the values according to Table 842.

Table 842 – NMEDomainQueueConfig.PreemptionMode

Value (hexadecimal)	Meaning
0x0	None
0x1	Preemptible queue
0x2	Express queue
Other	Reserved

Bit 16 – 39: NMEDomainQueueConfig.UnmaskTimeOffset
The bits shall be coded with the values according to Table 843.

Table 843 – NMEDomainQueueConfig.UnmaskTimeOffset

Data rate	Maximum gating cycle	UnmaskTimeOffset (hexadecimal)	Meaning
10 Mbit/s	8 ms	0x000000 – 0x7A1200	Offset from the start of the gating cycle in nanoseconds. The queue is visible for transmission selection with an offset from the start of the gating cycle.
100 Mbit/s	4 ms	0x000000 – 0x3D0900	
1 Gbit/s	1 ms	0x000000 – 0x0F4240	
> 1 Gbit/s	1 ms	0x000000 – 0x0F4240	
—	—	0x7A1201 – 0xFFFFFF	Reserved

Bit 40 – 63: NMEDomainQueueConfig.MaskTimeOffset
The bits shall be coded with the values according to Table 844.

Table 844 – NMEDomainQueueConfig.MaskTimeOffset

Data rate	Maximum gating cycle	MaskTimeOffset (hexadecimal)	Meaning
10 Mbit/s	8 ms	0x000000 – 0x7A1200	Offset from the start of the gating cycle in nanoseconds. The queue is visible for transmission selection with an offset from the start of the gating cycle.
100 Mbit/s	4 ms	0x000000 – 0x3D0900	
1 Gbit/s	1 ms	0x000000 – 0x0F4240	
> 1 Gbit/s	1 ms	0x000000 – 0x0F4240	
—	—	0x7A1201 – 0xFFFFFF	Reserved

5.2.12.4.3 Coding of the field PortQueueEgressRateLimiter

The coding of this field shall be according to 3.4.2.3.6 and the individual bits shall have the following meaning:

Bit 0 – 15: PortQueueEgressRateLimiter.CIR

The bits shall be coded with the values according to Table 845.

Table 845 – PortQueueEgressRateLimiter.CIR

Value (hexadecimal)	Meaning
0x0000	Used in case of no rate limiter
0x0001 – 0xFFFF	Committed information rate in 0,1 Mbit/s.

Bit 16 – 31: PortQueueEgressRateLimiter.CBS

The bits shall be coded with the values according to Table 846.

Table 846 – PortQueueEgressRateLimiter.CBS

Value (hexadecimal)	Meaning
0x0000	Used in case of no rate limiter
0x0001 – 0xFFFF	Committed burst size in octets.

Bit 32 – 39: PortQueueEgressRateLimiter.Envelope

The bits shall be coded with the values according to Table 847.

Table 847 – PortQueueEgressRateLimiter.Envelope

Value (hexadecimal)	Meaning
0x00	Used in case of no rate limiter
0x01	Best effort “envelope”
0x02 – 0xFF	Reserved

Bit 40 – 47: PortQueueEgressRateLimiter.Rank

The bits shall be coded with the values according to Table 848.

Table 848 – PortQueueEgressRateLimiter.Rank

Value (hexadecimal)	Meaning
0x00	Used in case of no boundary port.
0x01	Rank 1 used to address CIR^1, CBS^1 and CF^1
0x02	Rank 2 used to address CIR^2, CBS^2 and CF^2
0x03	Rank 3 used to address CIR^3, CBS^3 and CF^3
0x04	Rank 4 used to address CIR^4, CBS^4 and CF^4
0x05	Rank 5 used to address CIR^5, CBS^5 and CF^5
0x0006 – 0xFFFF	Reserved

Bit 48 – 55: PortQueueEgressRateLimiter.QueueID
The bits shall be coded with the values according to Table 849.

Table 849 – PortQueueEgressRateLimiter.QueueID

Value (hexadecimal)	Meaning
0x00 – 0x07	Identifier of the queue
0x08 – 0xFF	Reserved

Bit 56 – 63: PortQueueEgressRateLimiter.Reserved
The bits shall be coded according to Table 850.

Table 850 – PortQueueEgressRateLimiter.Reserved

Value (hexadecimal)	Meaning
0x00 – 0xFF	Reserved

5.2.12.4.4 Coding of the field CIMStationPortStatus

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0: CIMStationPortStatus.PreemptionStatus
The bits shall be coded with the values according to Table 851. The content of this field is provided by the QPSM state machine.

The content of this field is provided by the QPSM state machine.

Table 851 – CIMStationPortStatus.PreemptionStatus

Value (hexadecimal)	Meaning
0x00	Preemption is inactive for this link (local port to remote port).
0x01	Preemption is active for this link (local port to remote port).

Bit 1 – 3: CIMStationPortStatus.BoundaryPortStatus
The bits shall be coded with the values according to Table 852.

The content of this field is provided by the QPSM state machine.

Table 852 – CIMStationPortStatus.BoundaryPortStatus

Value (hexadecimal)	Meaning
0x00	No boundary port
0x01	Boundary port with Remapping 1 according to Table 403.
0x02	Boundary port with Remapping 2 according to Table 404.
0x03 – 0x07	Reserved

Bit 4 – 7: CIMStationPortStatus.Reserved

This field shall be set according to 3.4.2.2.

5.2.12.4.5 Coding of the field PortIngressRateLimiter

The coding of this field shall be according to 3.4.2.3.6 and the individual bits shall have the following meaning:

Bit 0 – 15: PortIngressRateLimiter.CIR

The bits shall be coded with the values according to Table 853.

Table 853 – PortIngressRateLimiter.CIR

Value (hexadecimal)	Meaning
0x0000	Used in case of no boundary port.
0x0001 – 0xFFFF	Committed information rate in 0,1 Mbit/s.

Bit 16 – 31: PortIngressRateLimiter.CBS

The bits shall be coded with the values according to Table 854.

Table 854 – PortIngressRateLimiter.CBS

Value (hexadecimal)	Meaning
0x0000	Used in case of no boundary port.
0x0001 – 0xFFFF	Committed burst size in octets.

Bit 32 – 47: PortIngressRateLimiter.Envelope

The bits shall be coded with the values according to Table 855.

Table 855 – PortIngressRateLimiter.Envelope

Value (hexadecimal)	Meaning
0x0000	Used in case of no boundary port.
0x0001	Best effort “envelope”
0x0002	RT_CLASS_X, RTA_CLASS_X „envelope“
0x0003 – 0xFFFF	Reserved

Bit 48 – 63: PortIngressRateLimiter.Rank

The bits shall be coded with the values according to Table 856.

Table 856 – PortIngressRateLimiter.Rank

Value (hexadecimal)	Meaning
0x0000	Used in case of no boundary port.
0x0001	Rank 1 used to address CIR^1, CBS^1 and CF^1
0x0002	Rank 2 used to address CIR^2, CBS^2 and CF^2
0x0003	Rank 3 used to address CIR^3, CBS^3 and CF^3
0x0004	Rank 4 used to address CIR^4, CBS^4 and CF^4
0x0005	Rank 5 used to address CIR^5, CBS^5 and CF^5
0x0006 – 0xFFFF	Reserved

5.2.12.4.6 Coding of the field GatingCycle

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 14: GatingCycle.SendClockFactor
This field shall be coded with values according to Table 638 and Table 639.

Bit 15: GatingCycle.Valid
This field shall be coded with values according to Table 857.

Table 857 – GatingCycle.Valid

Value (hexadecimal)	Meaning
0x00	Invalid, the field GatingCycle.SendClockFactor shall be set to zero.
0x01	Valid, the field GatingCycle.SendClockFactor shall be evaluated.

5.2.12.5 Coding section related to Physical Network Attributes

5.2.12.5.1 Coding of the field NumberOfQueues

This field shall be coded as data type Unsigned8 according to Table 858.

Table 858 – NumberOfQueues

Value (hexadecimal)	Meaning
0x06	The bridge supports six transmit queues at the port.
0x08	The bridge supports eight transmit queues at the port.
Other	Reserved

5.2.12.5.2 Coding of the field TransferTimeTX

This field shall be coded as data type Unsigned32 according to Table 859 and Figure 177 in nanoseconds.

Table 859 – TransferTimeTX

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x000F4240	Egress transfer time for the local interface of an end station
other	Reserved

5.2.12.5.3 Coding of the field TransferTimeRX

This field shall be coded as data type Unsigned32 according to Table 860 and Figure 177 in nanoseconds.

Table 860 – TransferTimeRX

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x000F4240	Ingress transfer time for the local interface of an end station
other	Reserved

5.2.12.5.4 Coding of the field PortCapabilities

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0: PortCapabilities.TimeAware

The bits shall be coded with the values according to Table 861.

Table 861 – PortCapabilities.TimeAware

Value (hexadecimal)	Meaning
0x00	This port is not usable within a time-aware system.
0x01	This port is usable within a time-aware system. The specified features of a port used in a NME domain are supported.

Bit 1: PortCapabilities.Preemption

The bits shall be coded with the values according to Table 862.

Table 862 – PortCapabilities.Preemption

Value (hexadecimal)	Meaning
0x00	Preemption is not supported at this port.
0x01	Preemption is supported at this port.

Bit 2: PortCapabilities.QueueMasking

The bits shall be coded with the values according to Table 863.

Table 863 – PortCapabilities.QueueMasking

Value (hexadecimal)	Meaning
0x00	Queue Masking is not supported at this port.
0x01	Queue Masking is supported at this port.

Bit 3 – 7: PortCapabilities.Reserved

This field shall be set according to 3.4.2.2.

5.2.12.5.5 Coding of the field ForwardingGroup

This field shall be coded as data type Unsigned8 according to Table 864.

Table 864 – ForwardingGroup

Value (hexadecimal)	Meaning
0x00 – 0xFF	Identifier of logical port grouping. Identifies ports with equal forwarding delay values.

5.2.12.5.6 Coding of the field ForwardingDelay

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 19: ForwardingDelay.Independent

The bits shall be coded with the values according to Table 865 in nanoseconds.

Table 865 – ForwardingDelay.Independent

Value (hexadecimal)	Meaning
0x00000	Reserved
0x00001 – 0xF4240	Frame length independent port to port forwarding delay value of a bridge used for calculation
other	Reserved

Bit 20 – 27: ForwardingDelay.Reserved

This field shall be set according to 3.4.2.2.

Bit 28 – 31: ForwardingDelay.Dependent

The bits shall be coded with the values according to Table 866.

Table 866 – ForwardingDelay.Dependent

Value (hexadecimal)	Meaning	Usage
0x0	Applies for Cut-Through forwarding	Additional port to port forwarding delay value of a bridge for each octet of a frame used for calculation
0x1	10 Mbit/s: 800 ns	
0x2	100 Mbit/s: 80 ns	
0x3	1 Gbit/s: 8 ns	
0x4	2,5 Gbit/s: 3 200 ps	
0x5	5 Gbit/s: 1 600 ps	
0x6	10 Gbit/s: 800 ps	
other	Reserved	—

5.2.12.5.7 Coding of the field MaxSupportedRecordSize

This field shall be coded as data type Unsigned32 according to Table 867.

Table 867 – MaxSupportedRecordSize

Value (hexadecimal)	Meaning
0x00000FE4 – 0x0000FFFF	Describes the maximum supported size of RecordDataWrite.
Other	Reserved

5.2.12.5.8 Coding of the field TrafficClassTranslationTable

5.2.12.5.8.1 General

End station application engineering is decoupled from the network engineering. Thus, the application defined traffic type need to be translated into network traffic classes.

Table 868 – Traffic classes

Index	Name	Abbreviation	PCP	VID
10	Stream High	HIGH / ISO	To be used value	To be used value
9	Stream High redundant	HIGH Red / ISO Red	To be used value	To be used value
8	Stream Low	LOW / CYC	To be used value	To be used value
7	Stream Low redundant	LOW Red / CYC Red	To be used value	To be used value
6	Stream RT	RT / A	To be used value	To be used value
5	Network Control	NW	To be used value	To be used value
4	Alarms and Events	EV	To be used value	To be used value
3	Connection Management	CO	To be used value	To be used value
2	Best effort High	BEH	To be used value	To be used value
1	Best effort Low	BEL	To be used value	To be used value

5.2.12.5.8.2 Coding of the field TrafficClassTranslateEntry

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 13: TrafficClassTranslateEntry.VID
This field shall be set according to Table 869.

Table 869 – TrafficClassTranslateEntry.VID

Value (hexadecimal)	Meaning
0x00	Reserved
0x01 – 0x0FFF	Allowed

Bit 14 – 15: TrafficClassTranslateEntry.Reserved1
This field shall be set to zero.

Bit 16 – 18: TrafficClassTranslateEntry.PCP
This field shall be set according to Table 870.

Table 870 – TrafficClassTranslateEntry.PCP

Value (hexadecimal)	Meaning
0x00 – 0x07	Allowed PCP values

Bit 19 – 31: TrafficClassTranslateEntry.Reserved2
This field shall be set to zero.

5.2.12.5.9 Coding of the end station transmit capabilities

5.2.12.5.9.1 Model

Figure 181 and Figure 182 show examples of the frame transmit characteristic of an end station component.

The visible IPG value at MDI may be dependent on the frame size of the successor and the frame size of the predecessor frame in a consecutive sequence of frames.

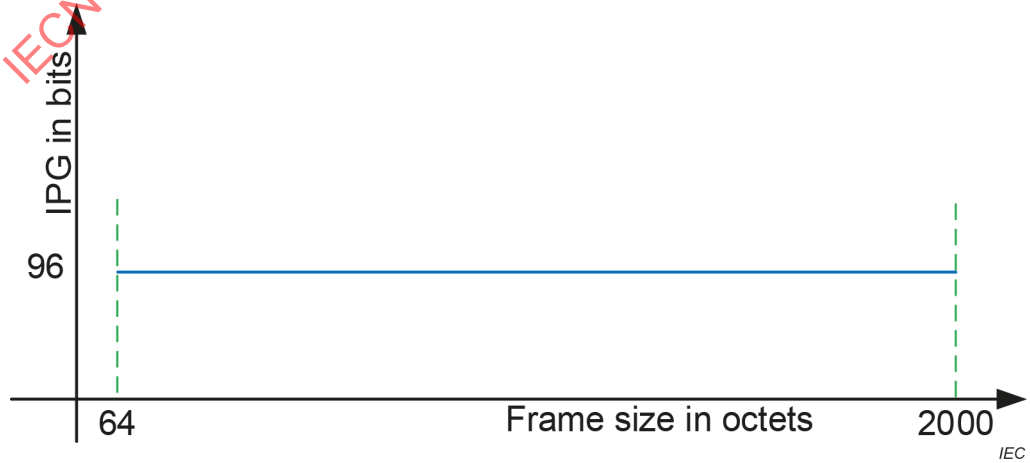


Figure 181 – Example IPG behavior of an ideal end station component in case of bursts

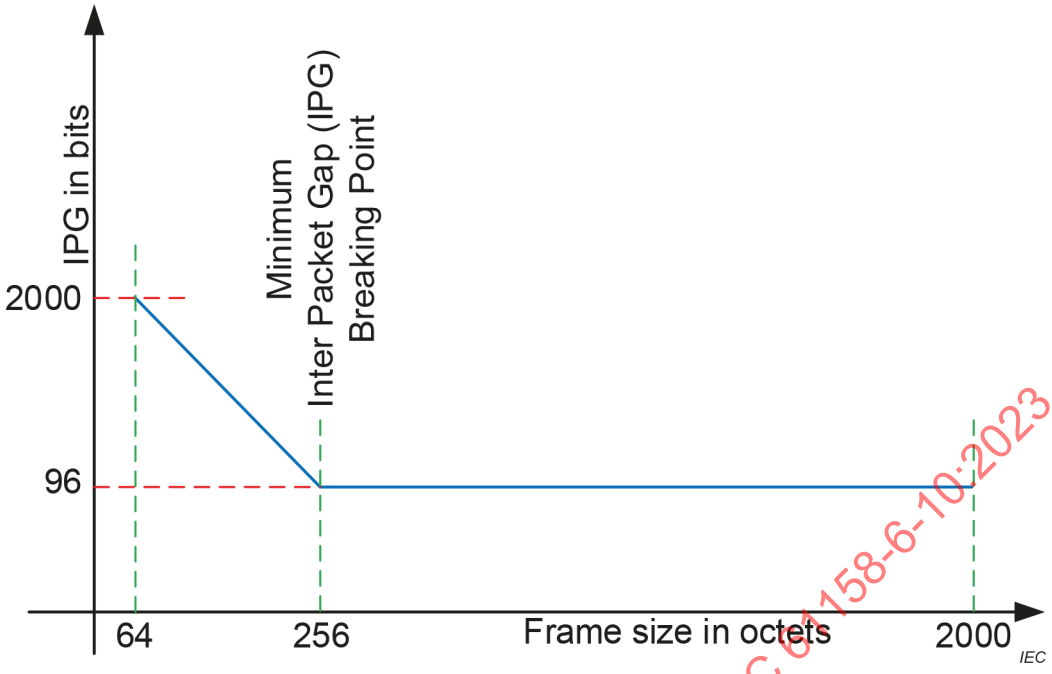


Figure 182 – Example IPG behavior of an end station component in case of bursts

5.2.12.5.9.2 Coding of the field MinIPGBreakingPoint

This field shall be coded as data type Unsigned16 with the values according to Table 871.

Table 871 – MinIPGBreakingPoint

Value (hexadecimal)	Meaning
0x0040 – 0x07D0	RT_CLASS_X Frame size in octets when the minimal IPG the last time achievable Example "256": Minimal IPG is no longer reachable if frames are shorter than 256 octets and the link speed of 1 Gbit/s Default: 0x0040
Other	Reserved

5.2.12.5.9.3 Coding of the field MinIPGFrameSize

This field shall be coded as data type Unsigned16 with the values according to Table 872.

Table 872 – MinIPGFrameSize

Value (hexadecimal)	Meaning
0x0028 – 0x07D0	RT_CLASS_X Reachable interframe gap in bit times for a sequence of minimum sized frames. Default: 0x0060
Other	Reserved

5.2.12.5.9.4 Coding of the field FrameSendOffsetDeviation

This field shall be coded as data type Unsigned16 with the values according to Table 873.

Table 873 – FrameSendOffsetDeviation

Value (hexadecimal)	Meaning
0x000A – 0x03E8	Maximum permissible deviation, later than expected, to the FrameSendOffset in nanoseconds. Default: 10 ns
Other	Reserved

5.2.12.5.9.5 Coding of the field SupportedBurstSize

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 15: SupportedBurstSize.Frames
This field shall be set according to Table 874.

Table 874 – SupportedBurstSize.Frames

Value (hexadecimal)	Meaning
0x00	Reserved
0x01 – 0x00FF	RT_CLASS_X Maximum number of frames per gating cycle Default: 0x00FF
0x0100 – 0xFFFF	Reserved

Bit 16 – 31: SupportedBurstSize.Octets
This field shall be set according to Table 875.

Table 875 – SupportedBurstSize.Octets

Value (hexadecimal)	Meaning
0x0000 – 0x003F	Reserved
0x0040 – 0x6400	RT_CLASS_X Maximum number of octets per gating cycle Default: 0x6400
0x6401 – 0xFFFF	Reserved

5.2.12.6 Coding section related to Physical Stream Path Data

5.2.12.6.1 Coding of the field FDBCommand

This field shall be coded as data type Unsigned8 with the values according to Table 876.

Table 876 – FDBCommand

Value (hexadecimal)	Meaning	Usage
0x01	AddStreamEntry	Add static stream entry to addressed FDB.
0x02	RemoveStreamEntry	Remove static stream entry from addressed FDB.
0x03	RemoveAllStreamEntries	Remove all static stream entries from addressed FDB.
Other	—	Reserved

5.2.12.6.2 Coding of the field StreamClass

This field shall be coded as data type Unsigned16 with the values according to Table 877.

Table 877 – StreamClass

Value (hexadecimal)	Meaning	VLANID
Other	Reserved	Reserved
0x0001	High	Stream High VID
0x0002	High Redundant	Stream High Redundant VID
0x0003	Low	Stream Low VID
0x0004	Low Redundant	Stream Low Redundant VID

5.2.12.7 Coding section related to Physical Sync Tree Data

5.2.12.7.1 Coding of the field SyncPortRole

This field shall be coded as data type Unsigned8 with the values according to Table 878.

Table 878 – SyncPortRole

Value (hexadecimal)	Meaning	Usage
0x00	The port is not part of the sync tree for this sync domain	—
0x01	Sync egress port for this sync domain	Working Clock/ Time Master port
0x02	Sync ingress port for this sync domain	Working Clock/ Time Slave port
others	Reserved	—

5.2.12.8 Coding section related to port statistic

5.2.12.8.1 Coding of the field CounterStatus

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0: CounterStatus.ifInOctets
This field shall be set according to Table 879.

Table 879 – CounterStatus.ifInOctets

Value (hexadecimal)	Meaning
0x00	The content of the field ifInOctets is valid
0x01	The content of the field ifInOctets is invalid. It shall be set to zero.

Bit 1: CounterStatus.ifOutOctets
This field shall be set according to Table 880.

Table 880 – CounterStatus.ifOutOctets

Value (hexadecimal)	Meaning
0x00	The content of the field ifOutOctets is valid
0x01	The content of the field ifOutOctets is invalid. It shall be set to zero.

Bit 2: CounterStatus.ifInDiscards
This field shall be set according to Table 881.

Table 881 – CounterStatus.ifInDiscards

Value (hexadecimal)	Meaning
0x00	The content of the field ifInDiscards is valid
0x01	The content of the field ifInDiscards is invalid. It shall be set to zero.

Bit 3: CounterStatus.ifOutDiscards
This field shall be set according to Table 882.

Table 882 – CounterStatus.ifOutDiscards

Value (hexadecimal)	Meaning
0x00	The content of the field ifOutDiscards is valid
0x01	The content of the field ifOutDiscards is invalid. It shall be set to zero.

Bit 4: CounterStatus.ifInErrors
This field shall be set according to Table 883.

Table 883 – CounterStatus.ifInErrors

Value (hexadecimal)	Meaning
0x00	The content of the field ifInErrors is valid
0x01	The content of the field ifInErrors is invalid. It shall be set to zero.

Bit 5: CounterStatus.ifOutErrors
This field shall be set according to Table 884.

Table 884 – CounterStatus.ifOutErrors

Value (hexadecimal)	Meaning
0x00	The content of the field ifOutErrors is valid
0x01	The content of the field ifOutErrors is invalid. It shall be set to zero.

Bit 6-15: CounterStatus.Reserved
This field shall be set according to Table 885.

Table 885 – CounterStatus.Reserved

Value (hexadecimal)	Meaning
0x00	Reserved

5.2.12.8.2 Coding of the field ifInOctets

This field shall be coded as data type Unsigned32 with the values according to IETF RFC 1213. The value zero shall be used if IETF RFC 1213 is not supported.

5.2.12.8.3 Coding of the field ifOutOctets

This field shall be coded as data type Unsigned32 with the values according to IETF RFC 1213. The value zero shall be used if IETF RFC 1213 is not supported.

5.2.12.8.4 Coding of the field ifInDiscards

This field shall be coded as data type Unsigned32 with the values according to IETF RFC 1213. The value zero shall be used if IETF RFC 1213 is not supported.

5.2.12.8.5 Coding of the field ifOutDiscards

This field shall be coded as data type Unsigned32 with the values according to IETF RFC 1213. The value zero shall be used if IETF RFC 1213 is not supported.

5.2.12.8.6 Coding of the field ifInErrors

This field shall be coded as data type Unsigned32 with the values according to IETF RFC 1213. The value zero shall be used if IETF RFC 1213 is not supported.

5.2.12.8.7 Coding of the field ifOutErrors

This field shall be coded as data type Unsigned32 with the values according to IETF RFC 1213. The value zero shall be used if IETF RFC 1213 is not supported.

5.2.12.9 Coding section related to fiber optics

5.2.12.9.1 Coding of the field VendorBlockType

This field shall be coded as data type Unsigned16 and set according to Table 886.

Table 886 – VendorBlockType

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Vendor specific

5.2.12.9.2 Coding of the field FiberOpticType

This field shall be coded as data type Unsigned32 and set according to Table 887.

Table 887 – FiberOpticType

Value (hexadecimal)	Meaning
0x00000000	No fiber type adjusted
0x00000001	9 µm single mode fiber
0x00000002	50 µm multi mode fiber
0x00000003	62,5 µm multi mode fiber
0x00000004	SI-POF [a], NA = 0,5
0x00000005	SI-PCF [a], NA = 0,36
0x00000006	LowNA-POF [a], NA = 0,3
0x00000007	GI-POF [a]
0x00000008	GI-PCF [a]
0x00000009 – 0x0000007F	Reserved
0x00000080 – 0x000000FF	Vendor specific
0x00000100 – 0xFFFFFFFF	Reserved
[a] See IEC 61158-2.	

5.2.12.9.3 Coding of the field FiberOpticCableType

This field shall be coded as data type Unsigned32. The coding shall be according to Table 888.

Table 888 – FiberOpticCableType

Value (hexadecimal)	Meaning
0x00000000	No cable specified
0x00000001	Inside/outside [a] cable, fixed installation
0x00000002	Inside/outside [a] cable, flexible installation
0x00000003	Outdoor cable, fixed installation
0x00000004 – 0xFFFFFFFF	Reserved
[a] This kind of cable may be used indoors for installation inside or outside of an electric cabinet.	

5.2.12.9.4 Coding of the field FiberOpticPowerBudgetType

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 30: FiberOpticPowerBudgetType.Value
This field shall be set according to Table 889.

Table 889 – FiberOpticPowerBudgetType.Value

Value (hexadecimal)	Meaning	Usage for PDPortFODataCheck	Usage for FiberOptic-PowerBudgetReal
0	0 dB	Mandatory for maintenance demanded	Measured value
0x0001 – 0x0013	0,1 dB to 1,9 dB	Optional	
0x0014	2 dB	Mandatory for maintenance required	
0x0015 – 0x03E7	2,1 dB to 99,9 dB	Optional	
0x03E8 – 0x7FFFFFFF	Reserved	—	—

Bit 31: FiberOpticPowerBudgetType.CheckEnable
This field shall be set according to Table 890.

Table 890 – FiberOpticPowerBudgetType.CheckEnable

Value (hexadecimal)	Meaning
0x0	OFF
0x1	ON Comparison value is FiberOpticPowerBudgetType.Value

5.2.12.9.5 Coding of the field FiberOpticManufacturerSpecific

This field shall be coded as data type OctetString.

5.2.12.10 Coding section related to SFP media modules

5.2.12.10.1 Coding of the field MaintenanceDemandedAdminStatus

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0: MaintenanceDemandedAdminStatus.Temperature
This field shall be set according to Table 891.

Table 891 – MaintenanceDemandedAdminStatus.Temperature

Value (hexadecimal)	Meaning
0x0	No check
0x1	Check for temperature high alarm

Bit 1: MaintenanceDemandedAdminStatus.TXBias
This field shall be set according to Table 892.

Table 892 – MaintenanceDemandedAdminStatus.TXBias

Value (hexadecimal)	Meaning
0x0	No check
0x1	Check for TXBias low/high alarm

Bit 2: MaintenanceDemandedAdminStatus.TXPower
This field shall be set according to Table 893.

Table 893 – MaintenanceDemandedAdminStatus.TXPower

Value (hexadecimal)	Meaning
0x0	No check
0x1	Check for TXPower low/high alarm

Bit 3: MaintenanceDemandedAdminStatus.RXPower
This field shall be set according to Table 894.

Table 894 – MaintenanceDemandedAdminStatus.RXPower

Value (hexadecimal)	Meaning
0x0	No check
0x1	Check for RXPower low/high alarm

Bit 4 – 31: MaintenanceDemandedAdminStatus.Reserved
This field shall be set according to Table 895.

Table 895 – MaintenanceDemandedAdminStatus.Reserved

Value (hexadecimal)	Meaning
0x0	Reserved

5.2.12.10.2 Coding of the field ErrorAdminStatus

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0: ErrorAdminStatus.TXFaultState
This field shall be set according to Table 896.

Table 896 – ErrorAdminStatus.TXFaultState

Value (hexadecimal)	Meaning
0x0	No check
0x1	Check for TXFaultState

Bit 1: ErrorAdminStatus.RXLossState
This field shall be set according to Table 897.

Table 897 – ErrorAdminStatus.RXLossState

Value (hexadecimal)	Meaning
0x0	No check
0x1	Check for RXLossState

Bit 2 – 31: ErrorAdminStatus.Reserved
This field shall be set according to Table 898.

Table 898 – ErrorAdminStatus.Reserved

Value (hexadecimal)	Meaning
0x0	Reserved

5.2.12.11 Coding section related to network components

5.2.12.11.1 Coding of the field NCDropBudgetType

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 30: NCDropBudgetType.Value
This field shall be set according to Table 899.

Table 899 – NCDropBudgetType.Value

Value (hexadecimal)	Meaning	Usage
0	Reserved	—
0x0001 – 0x0002	Number of dropped frames	Optional
0x0003	Number of dropped frames	Mandatory for maintenance required
0x0004 – 0x0009	Number of dropped frames	Optional
0x000A	Number of dropped frames	Mandatory for maintenance demanded
0x000B – 0x03E7	Number of dropped frames	Optional
0x03E8 – 0x7FFFFFFF	Reserved	—

Bit 31: NCDropBudgetType.CheckEnable
This field shall be set according to Table 900.

Table 900 – NCDropBudgetType.CheckEnable

Value (hexadecimal)	Meaning
0x0	OFF
0x1	ON Comparison value is NCDropBudgetType.Value

The checking of the dropped frames shall be done according to Figure 183 and Figure 184.

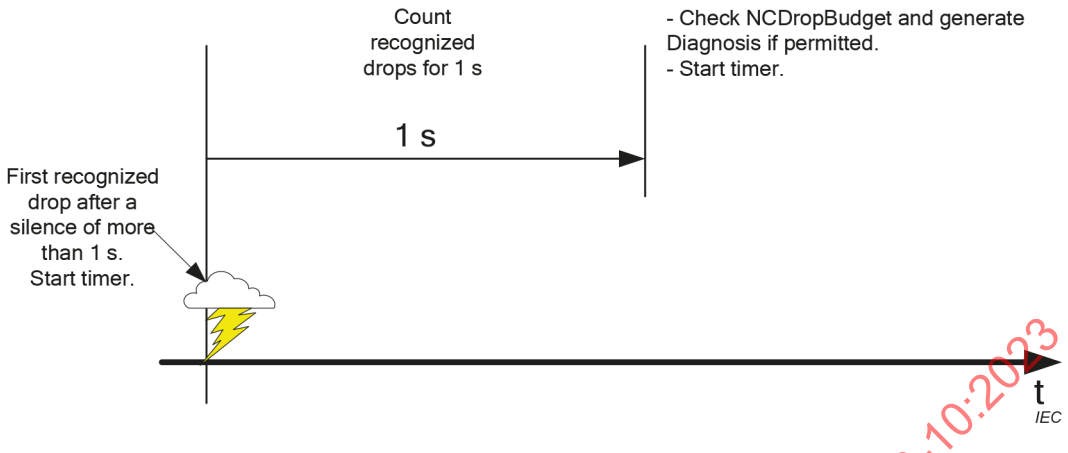


Figure 183 – Detection of dropped frames – appear

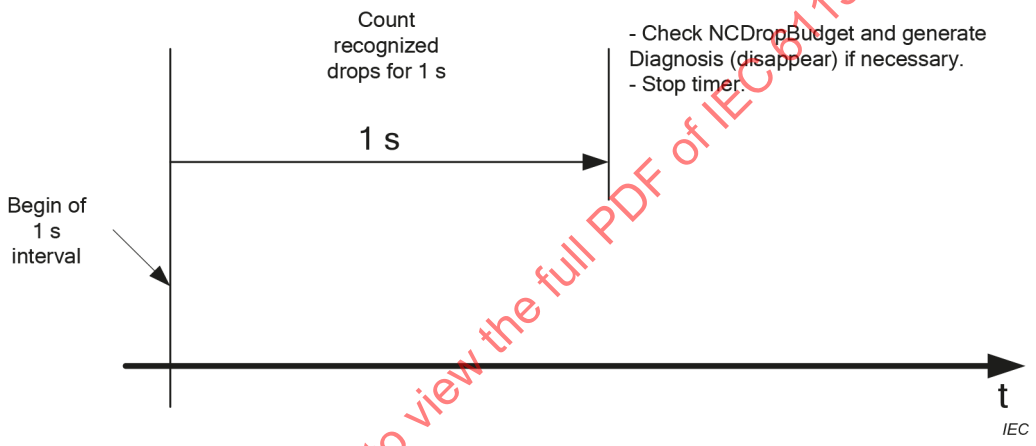


Figure 184 – Detection of dropped frames – disappear

5.2.12.12 Coding section related to Media Redundancy

5.2.12.12.1 General

5.2.12.12.1.1 Coding of the field MRP_Version

This field shall be coded as data type Unsigned16 and set according to Table 901.

Table 901 – MRP_Version

Value (hexadecimal)	Meaning
0x0001	Version 1
Other	Reserved

5.2.12.12.2 Coding section related to MRP Rings

5.2.12.12.2.1 Coding of the field MRP_RingState

This field shall be coded as data type Unsigned16 with the values according to Table 902.

Table 902 – MRP_RingState

Value (hexadecimal)	Meaning	Usage
0x0000	Ring open	Redundancy manager in Ring open state
0x0001	Ring closed	Redundancy manager in Ring closed state
0x0002 – 0xFFFF	Reserved	—

5.2.12.12.2.2 Coding of the field MRP_DomainUUID

This field shall be coded as data type UUID with the values according to Table 903.

The MRP_DomainUUID can be calculated according to Formula (74).

$$\text{MRP\_DomainUUID} = \text{MD5}(\text{MRP\_DomainName}) \quad (74)$$

where

MRP_DomainUUID is the MRP domain UUID

MD5 is the IETF RFC 6151 defined function to create a 128bit fingerprint

MRP_DomainName is the variable length input parameter MRP domain name

Table 903 – MRP_DomainUUID

Value (UUID)	Meaning	Usage
00000000-0000-0000-0000-000000000000	—	Reserved
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFE	UUID for media redundancy domain	Optional
FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF	Default UUID for media redundancy domain	Mandatory

5.2.12.12.2.3 Coding of the field MRP_LengthDomainName

This field shall be coded as data type Unsigned8 and set according to Table 904.

Table 904 – MRP_LengthDomainName

Value (hexadecimal)	Meaning
0x00	Reserved
0x01 – 0xF0	Allowed values
0xF1 – 0xFF	Reserved

5.2.12.12.2.4 Coding of the field MRP_DomainName

This field shall be coded as data type OctetString with 1 to 240 octets according to Table 905 and 4.3.1.4.16.

Table 905 – MRP_DomainName

Meaning	Usage
Name of the MRP Domain	—

NOTE The field MRP_DomainName is not terminated by zero.

5.2.12.12.2.5 Coding of the field MRP_Role

This field shall be coded as data type Unsigned16 and set according to Table 906.

Table 906 – MRP_Role

Value (hexadecimal)	Meaning
0x0000	Media Redundancy disabled
0x0001	Media Redundancy Client
0x0002	Media Redundancy Manager
0x0003	Media Redundancy Manager (Auto manager negotiation)
0x0004 – 0xFFFF	Reserved

5.2.12.12.2.6 Coding of the field MRP_Version

This field shall be coded as data type Unsigned16 and set according to Table 907.

Table 907 – MRP_Version

Value (hexadecimal)	Meaning
0x0001	Version 1
Other	Reserved

5.2.12.12.2.7 Coding of the field MRP_Prio

This field shall be coded as Unsigned16 and set according to Table 908.

Table 908 – MRP_Prio

Value (hexadecimal)	Meaning
0x0000	Highest priority redundancy manager
0x1000 – 0x7000	High priorities for redundancy manager
0x8000	Default priority for redundancy manager
0x9000 – 0x9FFF	High priorities for redundancy manager (Auto manager negotiation)
0xA000	Default priority for redundancy manager (Auto manager negotiation)
0xA001 – 0xF000	Low priorities for redundancy manager (Auto manager negotiation)
0xFFFF	Lowest priority for redundancy manager (Auto manager negotiation)
Other	Reserved

5.2.12.12.2.8 Coding of the field MRP_TOPchgT

This field shall be coded as data type Unsigned16 and set according to Table 909 with a time base of 10 ms.

It defines the common point of time that shall be used to invoke the Flush Filtering Data Base service.

Table 909 – MRP_TOPchgT

Value (decimal)	Meaning	Usage
0	0 ms	Clear filtering database (FDB table) immediately
1	10 ms	Mandatory
2 – 100	20 ms – 1 s	Optional
101 – 65 535	Reserved	—

5.2.12.12.2.9 Coding of the field MRP_TOPNRmax

This field shall be coded as data type Unsigned16 and set according to Table 910.

Table 910 – MRP_TOPNRmax

Value (decimal)	Meaning	Usage
0	Reserved	—
1	1 iteration	Optional
2	2 iterations	Optional
3	3 iterations	Mandatory (200 ms reconfiguration time)
4	4 iterations	Optional
5	5 iterations	Optional
6 – 65 535	Reserved	—

5.2.12.12.2.10 Coding of the field MRP_TSTshortT

This field shall be coded as data type Unsigned16 and set according to Table 911 with a time base of 1 ms.

Table 911 – MRP_TSTshortT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 9	“Value” × 1 ms 1 ms – 9 ms	Optional (short test interval)
10	“Value” × 1 ms 10 ms	Mandatory (200 ms reconfiguration time)
10 – 500	“Value” × 1 ms 10 ms – 500 ms	Optional (short test interval)
501 – 65 535	Reserved	—

5.2.12.12.2.11 Coding of the field MRP_TSTdefaultT

This field shall be coded as data type Unsigned16 and set according to Table 912 with a time base of 1 ms.

Table 912 – MRP_TSTdefaultT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 19	"Value" × 1 ms 1 ms – 19 ms	Optional (default test interval)
20	"Value" × 1 ms 20 ms	Mandatory (200 ms reconfiguration time)
21 – 1 000	"Value" × 1 ms 21 ms – 1 s	Optional (default test interval)
1 001 – 65 535	Reserved	—

5.2.12.12.2.12 Coding of the field MRP_TSTNRmax

This field shall be coded as data type Unsigned16 and set according to Table 913.

Table 913 – MRP_TSTNRmax

Value (decimal)	Meaning	Usage
0 – 1	Reserved	—
2	2 outstanding test indications cause ring failure	Optional
3	3 outstanding test indications cause ring failure	Mandatory (200 ms reconfiguration time)
4 – 10	4 – 10 outstanding test indications cause ring failure	Optional
11 – 65 535	Reserved	—

5.2.12.12.2.13 Coding of the field MRP_LNKdownT

This field shall be coded as data type Unsigned16 according to Table 914 with a time base of 1 ms.

Table 914 – MRP_LNKdownT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 19	"Value" × 1 ms 1 – 19 ms	Optional Link Down interval
20	"Value" × 1 ms 20 ms	Mandatory Link Down interval
21 – 1 000	"Value" × 1 ms 21 ms – 1 s	Optional Link Down interval
1 001 – 65 535	Reserved	—

5.2.12.12.2.14 Coding of the field MRP_LNKupT

This field shall be coded as data type Unsigned16 according to Table 915 with a time base of 1 ms.

Table 915 – MRP_LNKupT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 19	"Value" × 1 ms 1 – 19 ms	Optional Link Up interval
20	"Value" × 1 ms 20 ms	Mandatory Link Up interval
21 – 1 000	"Value" × 1 ms 21 ms – 1 s	Optional Link Up interval
1 001 – 65 535	Reserved	—

5.2.12.12.2.15 Coding of the field MRP_LNKNRmax

This field shall be coded as data type Unsigned16 according to Table 916.

Table 916 – MRP_LNKNRmax

Value (decimal)	Meaning	Usage
0	Reserved	—
1	1 iteration	Optional
2	2 iterations	Optional
3	3 iterations	Optional
4	4 iterations	Mandatory
5	5 iterations	Optional
6 – 65 535	Reserved	—

5.2.12.12.2.16 Coding of the field MRP_Check

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0: MRP_Check.MediaRedundancyManager
This field shall be set according to Table 917.

Table 917 – MRP_Check.MediaRedundancyManager

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Disable MediaRedundancyManager diagnosis
0x01	ON	Enable MediaRedundancyManager diagnosis (see Table 731)

Bit 1: MRP_Check.MRP_DomainUUID
This field shall be set according to Table 918.

Table 918 – MRP_Check.MRP_DomainUUID

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Disable the check of the MRP_DomainUUID vs. LLDP_PNIO_MRPPORTSTATUS
0x01	ON	Enable the check of the MRP_DomainUUID vs. LLDP_PNIO_MRPPORTSTATUS and the generation of "Peer MRP domain mismatch" (see Table 730)

Bit 2 – 23: MRP_Check.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 24 – 31: MRP_Check.Reserved_2
This field shall be set to zero.

5.2.12.12.2.17 Coding of the field MRP_NumberOfEntries

This field shall be coded as data type Unsigned8 according to Table 919.

Table 919 – MRP_NumberOfEntries

Value (decimal)	Meaning	Usage
1 – 127	Number of entries	—
Other	Reserved	—

5.2.12.12.2.18 Coding of the field MRP_Instance

This field shall be coded as data type Unsigned8 according to Table 920.

Table 920 – MRP_Instance

Value (decimal)	Meaning	Usage
0 – 126	Internal reference number of a MRP ring parameter set.	—
Other	Reserved	—

5.2.12.12.3 Coding section related to MRP Interconnections

5.2.12.12.3.1 Coding of the field MRPIC_LengthDomainName

This field shall be coded as data type Unsigned8 according to Table 921.

Table 921 – MRPIC_LengthDomainName

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
0x01 – 0xF0	Length of MRPIC_DomainName in octets	—
0xF1 – 0xFF	Reserved	—

5.2.12.12.3.2 Coding of the field MRPIC_DomainName

This field shall be coded as data type UnicodeString8 with a length of 1 to 240 octets according to Table 922.

Table 922 – MRPIC_DomainName

Meaning	Usage
Manufacturer specific defined name of the MRP Interconnection domain identified by MRPIC_DomainID	—

5.2.12.12.3.3 Coding of the field MRPIC_State

This field shall be coded as data type Unsigned16 according to Table 923.

Table 923 – MRPIC_State

Value (hexadecimal)	Meaning	Usage
0x00	MRP Interconnection open	—
0x01	MRP Interconnection closed	—
0x0002 – 0xFFFF	Reserved	—

5.2.12.12.3.4 Coding of the field MRPIC_Role

This field shall be coded as data type Unsigned16 according to Table 924.

Table 924 – MRPIC_Role

Value (hexadecimal)	Meaning	Usage
0x00	No role assigned	—
0x01	MRP Interconnection Client (MIC)	—
0x02	MRP Interconnection Manager (MIM)	—
0x0003 – 0xFFFF	Reserved	—

5.2.12.12.3.5 Coding of the field MRPIC_DomainID

This field shall be coded as data type Unsigned16 according to Table 925.

Table 925 – MRPIC_DomainID

Value (hexadecimal)	Meaning	Usage
0x0000	No MRP Interconnection Domain defined	—
0x0001 – 0xFFFF	Uniquely administered MRP Interconnection Domain identification	—

5.2.12.12.3.6 Coding of the field MRPIC_TOPchgT

This field shall be coded as data type Unsigned16 according to Table 926 with a time base of 10 ms.

It contains the interval for sending repetition of interconnection topology change frames for the MIM.

Table 926 – MRPIC_TOPchgT

Value (decimal)	Meaning	Usage
0	"Value" × 10 ms 0 ms	No repetition
1	"Value" × 10 ms 10 ms	Mandatory Sending repetition interval for interconnection topology change frame
2 – 100	"Value" × 10 ms 20 ms – 1 s	Optional Sending repetition interval for interconnection topology change frame
101 – 65 535	Reserved	—

5.2.12.12.3.7 Coding of the field MRPIC_TOPNRmax

This field shall be coded as data type Unsigned16 according to Table 927.

It contains the interval count which controls repeated transmission of interconnection topology change frames for the MIM.

Table 927 – MRPIC_TOPNRmax

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 2	1 to 2 iterations	Optional
3	3 iterations	Mandatory (200 ms reconfiguration time)
4 – 10	4 to 10 iterations	Optional
11 – 65 535	Reserved	—

Formula (75) defines the common point of time to invoke the Flush Filtering Data Base service.

$$FlushFDB = MRPIC\_TOPNRmax \times MRPIC\_TOPchgT \quad (75)$$

where

FlushFDB is the time after which the learned entries of the FDB are flushed after receiving the first interconnection topology change frame

MRPIC_TOPNRmax is the used repetition for the transmission of interconnection topology change frames

MRPIC_TOPchgT is the sending repetition interval for interconnection topology change frame

5.2.12.12.3.8 Coding of the field MRPIC_LinkStatusChangeT

This field shall be coded as data type Unsigned16 according to Table 928 with a time base of 1 ms.

It contains the interconnection Link Status poll time interval for the MIM.

Table 928 – MRPIC_LinkStatusChangeT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 19	"Value" × 1 ms 1 – 19 ms	Optional Link Status poll interval
20	"Value" × 1 ms 20 ms	Mandatory Link Status poll interval (200 ms reconfiguration time)
21 – 1 000	"Value" × 1 ms 21 ms – 1 s	Optional Link Status poll interval
1 001 – 65 535	Reserved	—

5.2.12.12.3.9 Coding of the field MRPIC_LinkStatusNRmax

This field shall be coded as data type Unsigned16 according to Table 929.

It contains the maximum number of outstanding responses to Link Status poll frames for the MIM. A higher value causes an interconnection failure.

Table 929 – MRPIC_LinkStatusNRmax

Value (decimal)	Meaning	Usage
0 – 7	Reserved	—
8	8 outstanding responses to polls	Mandatory (200 ms reconfiguration time)
9 – 65 535	9 – 65 535 outstanding responses to polls	Optional

5.2.12.12.3.10 Coding of the field MRPIC_LNKdownT

This field shall be coded as data type Unsigned16 according to Table 930 with a time base of 1 ms.

It contains the interval for sending link down Interconnection Link Change frames on ring ports and interconnection ports for the MIC.

Table 930 – MRPIC_LNKdownT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 19	"Value" × 1 ms 1 – 19 ms	Optional Link Down interval
20	"Value" × 1 ms 20 ms	Mandatory Link Down interval
21 – 1 000	"Value" × 1 ms 21 – 1 000 ms	Optional Link Down interval
1 001 – 65 535	Reserved	—

5.2.12.12.3.11 Coding of the field MRPIC_LNKupT

This field shall be coded as data type Unsigned16 according to Table 931 with a time base of 1 ms.

It contains the interval for sending link up Interconnection Link Change frames on ring ports and interconnection ports for the MIC.

Table 931 – MRPIC_LNKupT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 19	"Value" × 1 ms 1 – 19 ms	Optional Link Up interval
20	"Value" × 1 ms 20 ms	Mandatory Link Up interval
21 – 1 000	"Value" × 1 ms 21 – 1 000 ms	Optional Link Up interval
1 001 – 65 535	Reserved	—

5.2.12.12.3.12 Coding of the field MRPIC_LNKNRmax

This field shall be coded as data type Unsigned16 according to Table 932.

It contains the interconnection Link Change frame count which controls repeated transmission of interconnection Link up or Link down frames for the MIC.

Table 932 – MRPIC_LNKNRmax

Value (decimal)	Meaning	Usage
0	Reserved	—
1	1 iteration	Optional
2	2 iterations	Optional
3	3 iterations	Optional
4	4 iterations	Mandatory
5	5 iterations	Optional
6 – 100	6 – 100 iterations	Optional
101 – 65 535	Reserved	—

5.2.12.12.3.13 Coding of the field MRPIC_StartDelay

This field shall be coded as data type Unsigned16 according to Table 933 with a time base of 100 ms.

Table 933 – MRPIC_StartDelay

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
0x0001	"Value" × 100 ms 100 ms	Optional Startup delay
0x0002	"Value" × 100 ms 200 ms	Mandatory Startup delay
0x0003 – 0xEA60	"Value" × 100 ms	Optional Startup delay
0xEA61 – 0xFFFE	Reserved	—
0xFFFF	Wait for infinite time	Optional Startup delay

5.2.12.12.3.14 Coding of the field MRPIC_MICPosition

This field shall be coded as data type Unsigned16 according to Table 934.

It contains the position, either primary or secondary, a MIC is assigned to in the MRP interconnection topology.

Table 934 – MRPIC_MICPosition

Value (hexadecimal)	Meaning	Usage
0x0000	Primary	MRP Interconnection instance using this interconnection port is a primary MIC.
0x0001	Secondary	MRP Interconnection instance using this interconnection port is a secondary MIC.
Other	Reserved	—

5.2.12.12.3.15 Coding of the field MRPIC_Check

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0: MRPIC_Check.MIM
This field shall be set according to Table 935.

Table 935 – MRPIC_Check.MIM

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Disable MIM diagnosis
0x01	ON	Enable MIM diagnosis

Bit 1: MRPIC_Check.MRPIC_DomainID
This field shall be set according to Table 936.

Table 936 – MRPIC_Check.MRPIC_DomainID

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Disable the check of the MRPIC_DomainID vs. LLDP_PNIO_MRPICPORTSTATUS
0x01	ON	Enable the check of the MRPIC_DomainID vs. LLDP_PNIO_MRPICPORTSTATUS and its associated diagnosis

Bit 2 – 23: MRPIC_Check.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 24 – 31: MRPIC_Check.Reserved_2
This field shall be set to zero.

5.2.12.13 Coding section related to SNMPControl

5.2.12.13.1 Coding of the field SNMPControl

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: SNMPControl.SNMPControl
This field shall be set according to Table 937. Parallel SNMPv3 implementation shall not be influenced by this field.

Table 937 – SNMPControl.SNMPControl

Value (hexadecimal)	Meaning	Use
0x00	Disable SNMP	SNMPv1/v2 is disabled
0x01	Enable SNMP, Enable Read acces	Read access using SNMPv1/v2 to the in this document defined OIDs is enabled.
0x02	Enable SNMP, Enable Read and Write acces	Read and Write access using SNMPv1/v2 to the in this document defined OIDs is enabled.
0x03	Reserved	—

Bit 2 – 15: SNMPControl.Reserved_1
This field shall be set to zero.

5.2.12.13.2 Coding of the field CommunityNameLength

This field shall be coded as data type Unsigned8 with values according to Table 938.

Table 938 – CommunityNameLength

Value (hexadecimal)	Meaning
0x00	Shall be used within any read service. Shall be used in conjunction with SNMPControl if no writing is intended.
0x01 – 0xF0	Length of CommunityName in octets
0xF1 – 0xFF	Reserved

5.2.12.13.3 Coding of the field CommunityName

This field shall be coded as data type OctetString with 1 to 240 octets following the definition of Table 939 and the following syntax:

– Total length is 1 to 240 octets
– CommunityName consist of [a-zA-Z0-9.-]
– CommunityName do not start with [.-]
– CommunityName do not end with [.-]

Table 939 – CommunityName

Value [OctetString]	Meaning	Usage
public	SNMPv1/v2 community name	Default value for read-only community.
private	SNMPv1/v2 community name	Default value for read-write community.
any other OctetString	SNMPv1/v2 community name	User defined value

5.2.12.14 Coding section related to Electric Power Data

5.2.12.14.1 Coding of the field ElectricPowerDeviceVoltage

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 19: ElectricPowerDeviceVoltage.Voltage
This field shall be set according to Table 940.

Table 940 – ElectricPowerDeviceVoltage.Voltage

Value (hexadecimal)	Meaning	Use
0x00	Not supported	Not supported
0x00001 – 0xFFFFF	Supply voltage	Operational supply voltage Unit: mV Value range: 1 mV to 1,048575 kV Accuracy: ± 5 % of the measured value

Bit 20 – 27: ElectricPowerDeviceVoltage.Reserved
This field shall be set to zero.

Bit 28 – 31: ElectricPowerDeviceVoltage.Type
This field shall be set according to Table 941.

Table 941 – ElectricPowerDeviceVoltage.Type

Value (hexadecimal)	Meaning	Use
0x00	Reserved	—
0x01	Power from a 2-wire port	Power over DataLine according to MAUTypeExtension := APL
0x02	Power from a 4-wire port	Power over dedicated Power-In
0x03 – 0x0F	Reserved	—

5.2.12.14.2 Coding of the field ElectricPowerPortVoltage

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 19: ElectricPowerPortVoltage.Voltage
This field shall be set according to Table 942.

Table 942 – ElectricPowerPortVoltage.Voltage

Value (hexadecimal)	Meaning	Use
0x00000	Not supported	Not supported
0x00001 – 0xFFFFF	Supply voltage	Operational supply voltage Unit: mV Value range: 1 mV to 1,048575 kV Accuracy: ± 5 % of the measured value

Bit 20 – 27: ElectricPowerPortVoltage.Reserved
This field shall be set to zero.

Bit 28 – 31: ElectricPowerPortVoltage.Type
This field shall be set according to Table 943.

Table 943 – ElectricPowerPortVoltage.Type

Value (hexadecimal)	Meaning	Use
0x00	No power supply connected or power off	—
0x01	Power source Source of power supply for a connected device	Power over DataLine according to MAUTypeExtension := APL
0x02	Power load Source of power supply for this device	Power over DataLine according to MAUTypeExtension := APL
0x03 – 0x0F	Reserved	—

5.2.12.14.3 Coding of the field ElectricPowerPortCurrent

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 15: ElectricPowerPortCurrent.Current
This field shall be set according to Table 944.

Table 944 – ElectricPowerPortCurrent.Current

Value (hexadecimal)	Meaning	Use
0x0000	Not supported	Not supported

Value (hexadecimal)	Meaning	Use
0x0001 – 0xFFFF	Current	Operational current Power source: Current provided Power load: Current consumed Unit: mA Value range: 1 mA to 65,535 A Accuracy: ± 5 % of the measured value

Bit 16 – 31: ElectricPowerPortCurrent.CurrentLimit
This field shall be set according to Table 945.

Table 945 – ElectricPowerPortCurrent.CurrentLimit

Value (hexadecimal)	Meaning	Use
0x0000	Not supported	Not supported
0x0001 – 0xFFFF	Current	Power source: Effective current limit value Unit: mA Value range: 1 mA to 65,535 A Accuracy: ± 5 % of the measured value

5.2.13 Coding section related to Physical Sync Data

5.2.13.1 Coding of the field PTCPLengthSubdomainName

This field shall be coded as data type Unsigned8.

5.2.13.2 Coding of the field PTCPSubdomainName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.16.

NOTE The field PTCPSubdomainName is not terminated by zero.

5.2.13.3 Coding of the field SyncProperties

This field shall be coded according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: SyncProperties.Role
This field shall be coded with the values according to Table 946.

Table 946 – SyncProperties.Role

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
0x01	External sync	Working Clock Slave
0x02	Internal sync	Working Clock Master
0x03	Reserved	—

Bit 2 – 7: SyncProperties.Reserved
This field shall be set to zero.

Bit 8 – 12: SyncProperties.SyncID
This field shall be coded according to Table 947.

Table 947 – SyncProperties.SyncID

Value (hexadecimal)	Meaning	Usage
0x00	SyncID 0	Working Clock synchronization
Other	Reserved	—

Bit 13 – 15: SyncProperties.Reserved
This field shall be set to zero.

5.2.13.4 Coding of the field ReservedIntervalBegin

This field shall be coded as data type Unsigned32 with values according to Table 948, Table 950, and Figure 185. The time base shall be one nanosecond.

Table 948 – ReservedIntervalBegin

Value (hexadecimal)	Meaning
0x00000000	Mandatory
Other	Reserved

5.2.13.5 Coding of the field ReservedIntervalEnd

This field shall be coded as data type Unsigned32 with values according to Table 949, Table 950, and Figure 185. The time base shall be one nanosecond.

Table 949 – ReservedIntervalEnd

Value (hexadecimal)	Meaning
0x00000000	Mandatory
Other	Reserved

The dependencies between the field ReservedIntervalBegin and the field ReservedIntervalEnd are shown in Table 950.

Table 950 – Dependencies of ReservedIntervalBegin and ReservedIntervalEnd

Value (hexadecimal)	Meaning
ReservedIntervalBegin < ReservedIntervalEnd	An ORANGE period is defined
ReservedIntervalBegin = ReservedIntervalEnd = 0	Default
ReservedIntervalBegin > ReservedIntervalEnd	Not used

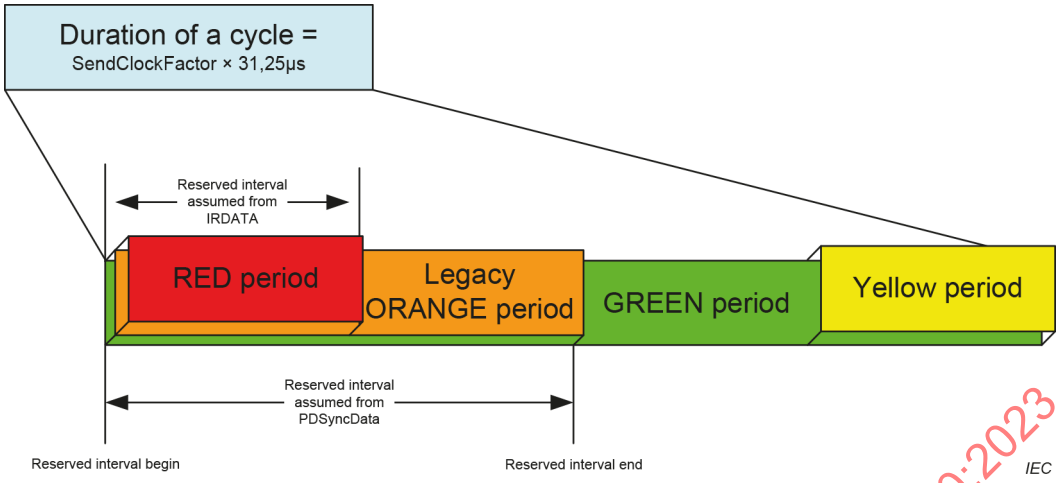


Figure 185 – Definition of the reserved interval

5.2.13.6 Coding of the field SyncSendFactor

This field shall be coded as data type Unsigned32 with values according to Table 951. The time base shall be 31,25 µs.

Table 951 – SyncSendFactor

Value (hexadecimal)	Meaning	SyncID (Clock)	SyncID (Time)
0x0000	Reserved	—	—
0x0001 – 0x03FF	Optional	—	—
0x03C0	Default	Mandatory (30 ms)	Optional (30 ms)
0x03C1 – 0x0C7F	Optional	—	—
0x0C80	Optional	—	Optional (100 ms)
0x0C81 – 0x176FF	Optional	—	—
0x17700	Default	—	Optional (3 s)
0x17701 – 0x4E1FF	Optional	—	—
0x4E200	Optional	—	Optional (10 s)
0x4E200 – 0xEA5FF	Optional	—	—
0xEA600	Default	—	Mandatory (30 s)
0x000EA601 – 0xA4CB7FFF	Optional	—	—
0xA4CB8000	Default	—	Optional (24 h)
0xA4CB8001 – 0xFFFFFFFF	Reserved	—	—

The SyncSendFactor may be rounded to full 10 ms before usage.

Each SyncProperties.SyncID shall require its own SyncSendFactor. The SyncSendInterval shall be calculated according to Formula (76).

$$\text{SyncSendInterval} = \text{SyncSendFactor} \times 31{,}25\ \mu s \quad (76)$$

where

SyncSendInterval is the sync send interval

SyncSendFactor is the factor for the calculation of the sync send interval

5.2.13.7 Coding of the field PTCPTimeoutFactor

This field shall be coded as data type Unsigned16. The time base is the value of the field SyncSendFactor. The value range shall be according to Table 952.

Table 952 – PTCPTimeoutFactor

Value (hexadecimal)	Meaning
0x0000	Disabled
0x0001 – 0x0002	Optional
0x0003 – 0x0005	Mandatory
0x0006	Default, mandatory
0x0007 – 0x000F	Mandatory
0x0010 – 0x01FF	Optional
0x0200 – 0xFFFF	Reserved

Each SyncProperties.SyncID shall require its own PTCPTimeoutFactor. The Timeout shall be calculated according to Formula (77).

$$\text{Timeout} = \text{PTCPTimeoutFactor} \times \text{SyncSendFactor} \times 31{,}25\ \mu\text{s} \quad (77)$$

where

Timeout is the timeout

PTCPTimeoutFactor is the PTCP timeout factor for the calculation of the timeout

SyncSendFactor is the factor for the calculation of the sync send interval

5.2.13.8 Coding of the field PTCPTakeoverTimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 953. The time base is the value of the field SyncSendFactor.

Table 953 – PTCPTakeoverTimeoutFactor

Value (hexadecimal)	Meaning
0x0000	Disabled
0x0001	Optional
0x0002	Mandatory Default for sync slaves
0x0003	Mandatory Default for sync masters
0x0004 – 0x000F	Mandatory
0x0010 – 0x01FF	Optional
0x0200 – 0xFFFF	Reserved

Each SyncProperties.SyncID shall require its own PTCPTakeoverTimeoutFactor. The Timeout shall be calculated according to Formula (78).

$$\text{Timeout} = \text{PTCPTakeoverTimeoutFactor} \times \text{SyncSendFactor} \times 31{,}25\ \mu s \quad (78)$$

where

Timeout	is the timeout
PTCPTakeoverTimeoutFactor	is the PTCP takeover factor for the calculation of the timeout
SyncSendFactor	is the factor for the calculation of the sync send interval
31,25 µs	is the time base for the calculation

5.2.13.9 Coding of the field PTCPMasterStartupTime

This field shall be coded as data type Unsigned16 with the values according to Table 954. The time base shall be one second.

Table 954 – PTCPMasterStartupTime

Value (hexadecimal)	Meaning
0x0000	Disabled
0x0001 – 0x0004	Optional
0x0005 – 0x003B	Mandatory
0x003C	Default, mandatory
0x003D – 0x012C	Optional
0x012D – 0xFFFF	Reserved

Each SyncProperties.SyncID shall require its own PTCPMasterStartupTime.

5.2.13.10 Coding of the field PLLWindow

This field shall be coded as data type Unsigned32 with the values according to Table 955. The time base shall be one nanosecond.

Table 955 – PLLWindow

Value (hexadecimal)	Meaning	SyncID := 0 (Working Clock)
0x00000000	Disabled	—
0x00000001 – 0x000003E7	Optional	—
0x000003E8	Default	Mandatory (1 µs)
0x000003E9 – 0x0000270F	Optional	—
0x00002710	Default	Optional (10 µs)
0x00002710 – 0x000F423F	Optional	—
0x000F4240	Default	Optional (1 ms)
0x000F423F – 0x0098967F	Optional	—
0x00989680	Default	Optional (10 ms)
0x00989681 – 0xFFFFFFFF	Reserved	—

Each SyncProperties.SyncID shall require its own PLLWindow.

The definition of the PLLWindow is shown in Figure 186 and Figure 187.

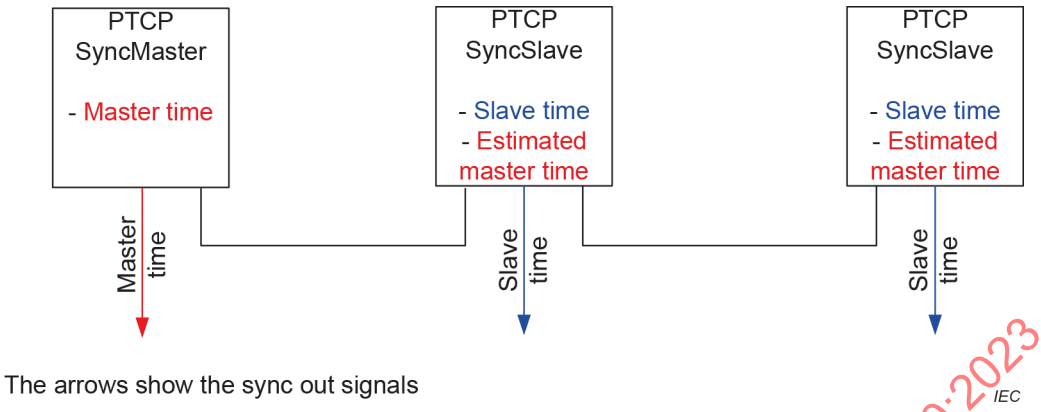


Figure 186 – Toplevel view of the PLL window

NOTE 1 The SyncSlave estimates the master time by means of PTCP. It controls its slave time to follow the master time, but in actual fact it follows the estimated master time. Because of the noisy environment the estimated master time is more or less equal to the master time.

NOTE 2 The external measurement of the PLLWindow is done between master time and slave time. The internal measurement of a SyncSlave is done between estimated master time and slave time.

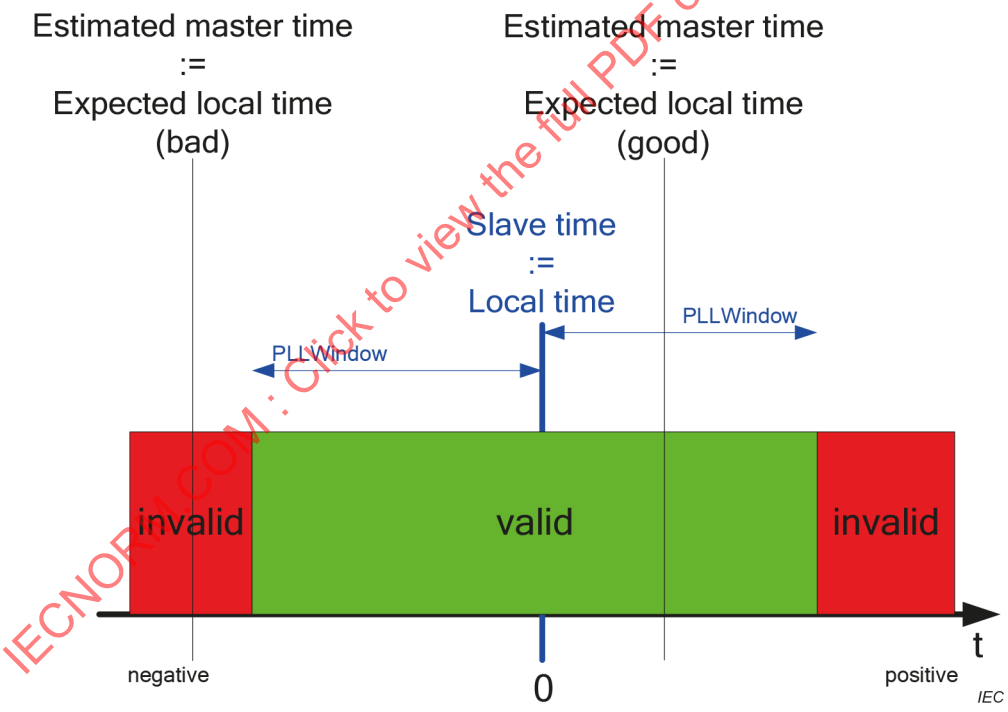


Figure 187 – Definition of PLL window

5.2.14 Coding section related to Physical Time Data

5.2.14.1 Coding of the field TimeDomainNameUUID

This field shall be coded as data type UUID with the values according to Table 956.

The TimeDomainUUID shall be calculated according to Formula (79).

$$TimeDomainUUID = MD5(TimeDomainName) \quad (79)$$

where

TimeDomainUUID is the Time Domain UUID

MD5 is the IETF RFC 6151 defined function to create a 128 bit fingerprint

TimeDomainName is the variable length input parameter Time Domain name

Table 956 – TimeDomainUUID

Value (UUID)	Meaning
00000000-0000-0000-0000-000000000000	No TimeDomain assigned
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF	UUID identifying a TimeDomain

5.2.14.2 Coding of the field TimeDomainNameLength

This field shall be coded as data type Unsigned8.

5.2.14.3 Coding of the field TimeDomainName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.16.

NOTE The field TimeDomainName is not terminated by zero.

5.2.14.4 Coding of the field TimeDomainNumber

This field shall be coded as data type Unsigned16 with the values according to Table 957.

Table 957 – TimeDomainNumber

Value (hexadecimal)	Meaning [Timescale]	Usage
0x0000	Global Time	Used for the synchronization of Global Time timescale.
0x0001	Global Time Redundant	Used for the synchronization of Global Time timescale together with a hot standby synchronization master.
0x0020	Working Clock	Used for the synchronization of Working Clock timescale.
0x0021	Working Clock Redundant	Used for the synchronization of Working Clock timescale together with a hot standby synchronization master.
Other	—	Reserved

5.2.14.5 Coding of the field TimePLLWindow

This field shall be coded as data type Unsigned32 with the values according to Table 958. The time base shall be one nanosecond.

Table 958 – TimePLLWindow

Value (hexadecimal)	Meaning	SyncID (Global Time)	SyncID (Working Clock)
0x00000000	Disabled	—	—
0x000003E8	Default	Optional (1 µs)	Mandatory (1 µs)
0x00002710	Default	Optional (10 µs)	Optional (10 µs)
0x000186A0	Default	Mandatory (100 µs)	Optional (100 µs)
0x000F4240	Default	Optional (1 ms)	Optional (1 ms)
0x00989680	Default	Optional (10 ms)	Optional (10 ms)
other	Reserved	—	—

The definition of the TimePLLWindow is shown in Figure 188 and Figure 189.

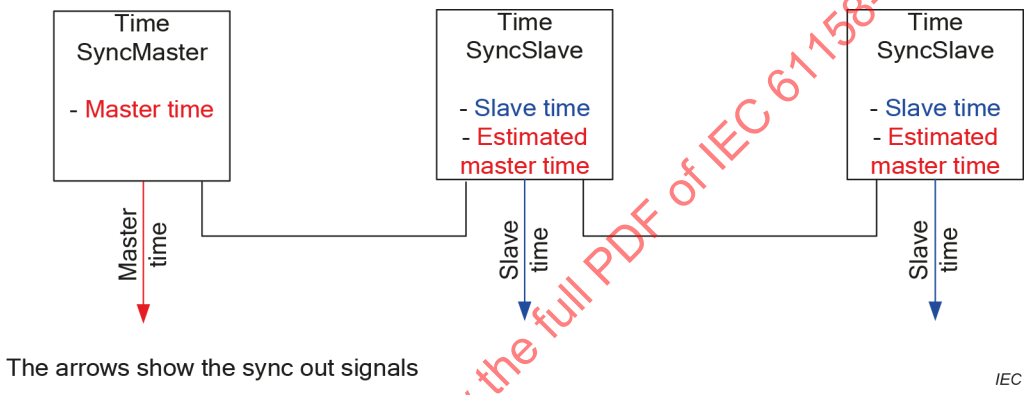


Figure 188 – Toplevel view of the time PLL window

NOTE 1 The SyncSlave estimates the master time by means of IEEE Std 802.1AS. It controls its slave time to follow the master time, but in actual fact it follows the estimated master time. Because of the noisy environment the estimated master time is more or less equal to the master time.

NOTE 2 The external measurement of the TimePLLWindow is done between master time and slave time. The internal measurement of a SyncSlave is done between estimated master time and slave time.

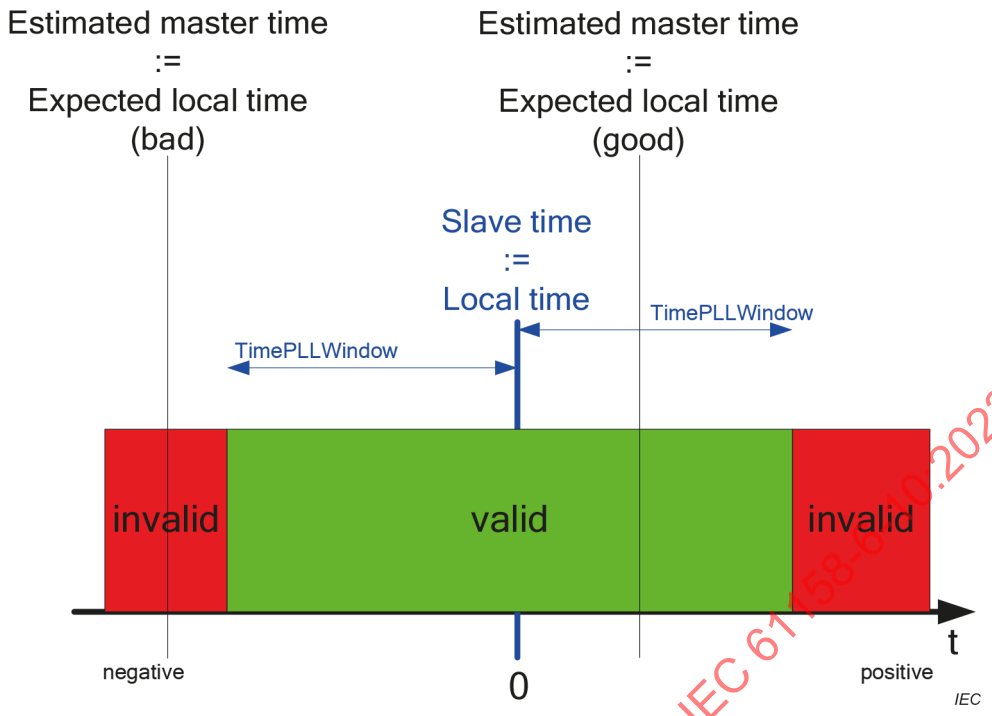


Figure 189 – Definition of time PLL window

5.2.14.6 Coding of the field TimeMasterPriority1

This field shall be coded as Unsigned8 with the value according to Table 959.

Table 959 – TimeMasterPriority1

Value (hexadecimal)	Usage	Meaning
0x00	Reserved	—
0x01 – 0x7F	Master	Optional
0x80	Master	Default for the role Time Master
0x81 – 0xFE	Master	Optional
0xFF	Slave	Default for the role Time slave

5.2.14.7 Coding of the field TimeMasterPriority2

This field shall be coded as Unsigned8 with the value according to Table 960.

Table 960 – TimeMasterPriority2

Value (hexadecimal)	Usage	Meaning
0x00 – 0xF7	Reserved	—
0xF8	Default	Default
0xF9 – 0xFF	Reserved	—

5.2.14.8 Coding of the field MessageIntervalFactor

This field shall be coded as data type Unsigned32 with values according to Table 961. The time base shall be 31,25 µs.

Table 961 – MessageIntervalFactor

Value (hexadecimal)	Meaning	TimeDomainNumber (WorkingClock)	TimeDomainNumber (GlobalTime)
0x00000000	Reserved	—	—
0x000003E8	Default	Mandatory (31,25 ms)	—
0x00000FA0	Default	—	Mandatory (125 ms)
Others	Optional	—	—

The MessageIntervalFactor may be rounded to full 10 ms before usage.

Each TimeDomainNumber shall require its own MessageIntervalFactor. The MessageInterval shall be calculated according to Formula (80).

$$\text{MessageInterval} = \text{MessageIntervalFactor} \times 31{,}25\ \mu\text{s} \quad (80)$$

where

MessageInterval is the sync send interval

MessageIntervalFactor is the factor for the calculation of the sync send interval

5.2.14.9 Coding of the field MessageTimeoutFactor

This field shall be coded as data type Unsigned16. The time base is the value of the field MessageIntervalFactor. The value range shall be according to Table 962.

Table 962 – MessageTimeoutFactor

Value (hexadecimal)	Meaning
0x0000	Disabled
0x0001 – 0x0002	Optional
0x0003 – 0x0005	Mandatory
0x0006	Default, mandatory
0x0007 – 0x000F	Mandatory
0x0010 – 0x01FF	Optional
0x0200 – 0xFFFF	Reserved

Each TimeDomainNumber shall require its own MessageTimeoutFactor. The Timeout shall be calculated according to Formula (81).

$$\text{Timeout} = \text{MessageTimeoutFactor} \times \text{MessageIntervalFactor} \times 31{,}25\ \mu\text{s} \quad (81)$$

where

Timeout is the timeout

MessageTimeoutFactor is the message timeout factor for the calculation of the timeout

MessageIntervalFactor is the factor for the calculation of the sync message send interval

5.2.14.10 Coding of the field TimeSyncProperties

This field shall be coded according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: TimeSyncProperties.Role
This field shall be coded with the values according to Table 963.

Table 963 – TimeSyncProperties.Role

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
0x01	External sync	Working Clock / Time Slave
0x02	Internal sync	Working Clock / Time Master
0x03	Reserved	—

Bit 2 – 15: TimeSyncProperties.Reserved
This field shall be set to zero.

5.2.15 Coding section related to Isochrone Mode Data

5.2.15.1 Coding of the field TimeIOBase

This field shall be coded as data type Unsigned32 with values according to Table 964. The time base shall be one nanosecond.

Table 964 – TimeIOBase

Value (decimal)	Meaning
1 – 32 000 000	Optional
Other	Reserved

5.2.15.2 Coding of the field TimeDataCycle

This field shall be coded as data type Unsigned16 with values according to Table 965. The time base shall be 31,25 µs.

Table 965 – TimeDataCycle

Value (hexadecimal)	Meaning	Comment
0x0000	Reserved	—
0x0001	Optional	Minimum time of 31,25 µs
0x0002 – 0x03FF	Optional	…
0x0400	Optional	Maximum time of 32 ms.
0x0401 – 0xFFFF	Reserved	—

The calculation of the TimeDC shall be done by Formula (82).

$$\text{TimeDC} = \text{TimeDataCycle} \times 31{,}25\ \mu\text{s} \quad (82)$$

where

TimeDC	is the time for the isochronous data cycle
TimeDataCycle	is the factor for the TimeDC
31,25 µs	is the time base for the calculation

5.2.15.3 Coding of the field TimeIOInput

This field shall be coded as data type Unsigned32 with values according to Table 966. The time base shall be one nanosecond.

Table 966 – TimeIOInput

Value (decimal)	Meaning
0 – 32 000 000	Optional
Other	Reserved

5.2.15.4 Coding of the field TimeIOOutput

This field shall be coded as data type Unsigned32 with values according to Table 967. The time base shall be one nanosecond.

Table 967 – TimeIOOutput

Value (decimal)	Meaning
0 – 32 000 000	Optional
Other	Reserved

5.2.15.5 Coding of the field TimeIOInputValid

This field shall be coded as data type Unsigned32 with values according to Table 968. The time base shall be one nanosecond.

Table 968 – TimeIOInputValid

Value (decimal)	Meaning
0 – 32 000 000	Optional
Other	Reserved

5.2.15.6 Coding of the field TimeIOOutputValid

This field shall be coded as data type Unsigned32 with values according to Table 969. The time base shall be one nanosecond.

Table 969 – TimeIOOutputValid

Value (decimal)	Meaning
0 – 32 000 000	Optional
Other	Reserved

5.2.15.7 Coding of the field ControllerApplicationCycleFactor

This field shall be coded as data type Unsigned16 with values according to Table 970.

Table 970 – ControllerApplicationCycleFactor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x0400	Optional
0x0401 – 0xFFFF	Reserved

5.2.16 Coding section related to fast startup

5.2.16.1 Coding of the field FSHelloMode

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 1: FSHelloMode.Mode
This field shall be set according to Table 971.

Table 971 – FSHelloMode.Mode

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	Send DCP_Hello.req on LinkUp	DCP_Hello requests shall be sent as soon as a LinkUp is detected at one port.
0x02	Send DCP_Hello.req on LinkUp after HelloDelay	DCP_Hello requests shall be sent after the HelloDelay when a LinkUp is detected at one port.
0x03	Reserved	Reserved

Bit 2 – 23: FSHelloMode.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 24 – 31: FSHelloMode.Reserved_2
This field shall be set to zero.

5.2.16.2 Coding of the field FSHelloInterval

This field shall be coded as data type Unsigned32. The coding shall be according to Table 972.

Table 972 – FSHelloInterval

Value (hexadecimal)	Meaning	Usage
0x00000001E	30 ms Wait for this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	Default
0x000000032	50 ms Wait for this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
0x000000064	100 ms Wait for this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
0x00000012C	300 ms Wait for this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
0x0000001F4	500 ms Wait for this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
0x0000003E8	1 000 ms Wait for this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
Other	Reserved	—

5.2.16.3 Coding of the field FSHelloRetry

This field shall be coded as data type Unsigned32. The coding shall be according to Table 973.

Table 973 – FSHelloRetry

Value (hexadecimal)	Meaning	Usage
0x00000000	Reserved	—
0x00000001 – 0x00000002	Number of retransmission of the Hello.req	—
0x00000003	Number of retransmission of the Hello.req	Default
0x00000004 – 0x0000000F	Number of retransmission of the Hello.req	—
0x00000010 – 0xFFFFFFFF	Reserved	—

5.2.16.4 Coding of the field FSHelloDelay

This field shall be coded as data type Unsigned32. The coding shall be according to Table 974.

Table 974 – FSHelloDelay

Value (hexadecimal)	Meaning	Usage
0x000000000	OFF	Default
0x000000032	50 ms Wait for this time after the first LinkUp.ind before conveying a DCP_Hello.req	—
0x000000064	100 ms Wait for this time after the first LinkUp.ind before conveying a DCP_Hello.req	—
0x0000001F4	500 ms Wait for this time after the first LinkUp.ind before conveying a DCP_Hello.req	—
0x0000003E8	1 000 ms Wait for this time after the first LinkUp.ind before conveying a DCP_Hello.req	—
Other	Reserved	—

5.2.16.5 Coding of the field FSParameterMode

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 1: FSParameterMode.Mode
This field shall be set according to Table 975.

Table 975 – FSParameterMode.Mode

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	ON	—
0x02	Reserved	—
0x03	Reserved	—

Bit 2 – 23: FSParameterMode.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 24 – 31: FSParameterMode.Reserved_2
This field shall be set to zero.

5.2.16.6 Coding of the field FSParameterUUID

This field shall be coded as data type UUID with values according to Table 976.

Table 976 – FSParameterUUID

Value (UUID)	Meaning
00000000-0000-0000-0000-000000000000	Reserved
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF	UUID for the record data (including physical device data, ...) delivered between IODConnectRes and IODControlReq.

5.2.17 Coding section related to DFP

5.2.17.1 Coding of the field NumberOfSubframeBlocks

This field shall be coded as data type Unsigned16 with values according to Table 977.

Table 977 – NumberOfSubframeBlocks

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x00FF	Number of the following SubframeBlocks
0x0100 – 0xFFFF	Reserved

5.2.17.2 Coding of the field SFIOCRProperties

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0 – 7: SFIOCRProperties.DistributedWatchDogFactor
This field shall be set according to Table 978.

Table 978 – SFIOCRProperties.DistributedWatchDogFactor

Value (hexadecimal)	Meaning	Description
0x00 – 0x02	Reserved	—
0x03 – 0x1F	Mandatory	An expiration notification should be signaled to the application as a hint for a possible communication disturbance.
0x20 – 0xFF	Reserved	—

The time base is the SendClockFactor multiplied by the ReductionRatio of the monitored provider.

The DistributedWatchDogTime shall be calculated according to Formula (83).

$$\text{DistributedWatchDogTime} = \text{DistributedWatchDogFactor} \times \text{SendClockFactor} \times \text{ReductionRatio} \times 31{,}25\ \mu\text{s} \quad (83)$$

where

DistributedWatchDogTime	is the distributed watchdog time
DistributedWatchDogFactor	is the factor for the calculation of the distributed watchdog
SendClockFactor	is the send clock factor
ReductionRatio	is the reduction ratio

Bit 8 – 15: SFIOCRProperties.RestartFactorForDistributedWD
This field shall be set according to Table 979.

Table 979 – SFIOCRProperties.RestartFactorForDistributedWD

Value (hexadecimal)	Meaning	Description
0x00	Mandatory	No restart delay necessary
0x01 – 0x09	Optional	Less than 1 s restart delay
0x0A – 0x50	Mandatory	1 s to 8 s restart delay
0x51 – 0xFF	Optional	More than 8 s restart delay

The time base shall be 100 ms. The RestartForDistributedWDTime shall be calculated according to Formula (84).

$$\text{RestartForDistributedWDTime} = \text{RestartFactorForDistributedWDFactor} \times 100\ \text{ms} \quad (84)$$

where

RestartForDistributedWDTime is the restart for distributed watchdog time

RestartFactorForDistributedWDFactor is the factor for the calculation of the restart for distributed watchdog time

NOTE The RestartForDistributedWDTime is defined as minimum time having an accuracy between '-0' and '+1 s'.

Bit 16 – 23: SFIOCRProperties.DFPMode
This field shall be set according to Table 980.

Table 980 – SFIOCRProperties.DFPMode

Value (hexadecimal)	Meaning	Description
0x00	Mandatory	End node mode
0x01 – 0x3F	Mandatory	Concatenating mode Own Subframe position
0x40 – 0xFF	Optional	Concatenating mode Own Subframe position

Bit 24 – 27: SFIOCRProperties.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 28: SFIOCRProperties.Reserved_2
This field shall be set to zero.

Bit 29: SFIOCRProperties.DFPDirection
This field shall be set according to Table 981.

Table 981 – SFIOCRProperties.DFPDirection

Value (hexadecimal)	Meaning	Description
0x00	Mandatory	This Frame is intended to be handled by the DFP_RELAY_INBOUND state machine.
0x01	Mandatory	This Frame is intended to be handled by the DFP_RELAY_OUTBOUND state machine.

Bit 30: SFIOCRProperties.DFPRedundantPathLayout
This field shall be set according to Table 982.

Table 982 – SFIOCRProperties.DFPRedundantPathLayout

Value (hexadecimal)	Meaning	Description
0x00	Mandatory	The Frame for the redundant path, coded by the FrameID with the least significant bit set to 1, shall contain the ordering shown by SubframeData.
0x01	Optional	The Frame for the redundant path, coded by the FrameID with the least significant bit set to 1, shall contain the inverse ordering of the SubframeData.

Bit 31: SFIOCRProperties.SFCRC16
This field shall be set according to Table 983.

Table 983 – SFIOCRProperties.SFCRC16

Value (hexadecimal)	Description
0x00	SFCRC16 and SFCycleCounter shall be created or set to zero by the sender and not checked by the receiver.
0x01	SFCRC16 and SFCycleCounter shall be created by the sender and checked by the receiver.

5.2.17.3 Coding of the field SubframeData

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning.

NOTE The SFCRC16 of the frame header and the SFEndDelimiter are implicit. See CSF_SDU in 4.7.1.2.

Bit 0 – 6: SubframeData.Position
This field shall be set according to Table 984.

Table 984 – SubframeData.Position

Value (hexadecimal)	Meaning
0x00	Reserved
0x01 – 0x7F	SFPosition.Position of the Subframe

Bit 7: SubframeData.Reserved_1
This field shall be set according to 3.4.2.2.

Bit 8 – 15: SubframeData.DataLength
This field shall be set according to Table 985.

Table 985 – SubframeData.DataLength

Value (hexadecimal)	Meaning
0x00	Reserved
0x01 – 0xFF	Number of octets of the C_SDU of the SUBFRAME

Bit 16 – 31: SubframeData.Reserved_2
This field shall be set according to 3.4.2.2.

5.2.17.4 Frame late error

The checking of the DFP frame late error shall be done according to 4.12.11.5.5.4. Figure 190 and Table 986 show the behavior as an overview.

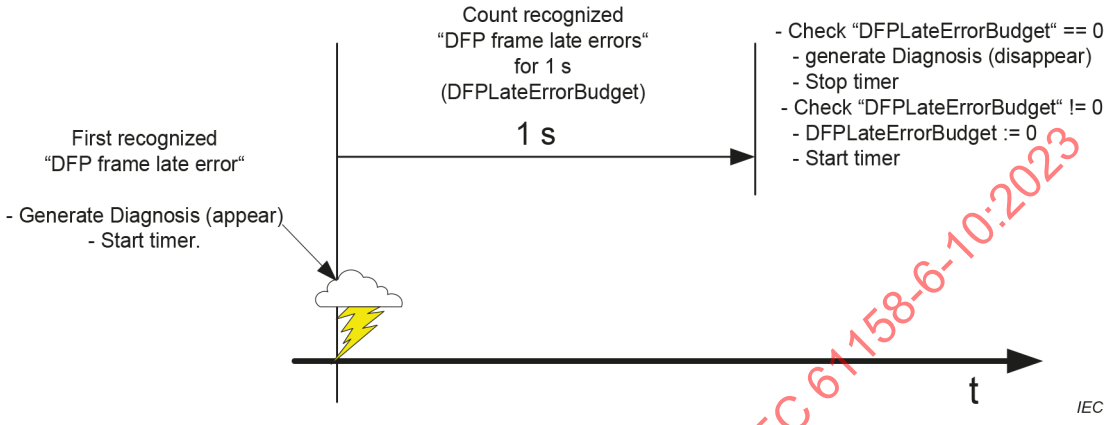


Figure 190 – Detection of DFP late error – appear and disappear

Table 986 – Event function table

Function name	Operations
Appear	Alarm Priority := ALARM_HIGH or ALARM_LOW Alarm Type:= Dynamic Frame Packing problem notification User Structure Identifier := 0x8002 ChannelNumber := 0x8000 ChannelProperties.Specifier := Appears ChannelProperties.Maintenance := MaintenanceDemanded ChannelErrorType := “Dynamic frame packing function mismatch” ExtChannelErrorType := “Frame late error” (see Formula (65)) Alarm Notification.req(AREP, API, Alarm Priority, Alarm Type, Slot Number, Subslot Number, Alarm Specifier, Module Ident Number, Submodule Ident Number, Alarm Item)
Disappear	Alarm Priority := ALARM_HIGH or ALARM_LOW Alarm Type:= Dynamic Frame Packing problem notification User Structure Identifier := 0x8002 ChannelNumber := 0x8000 ChannelProperties.Specifier := Disappears [a] ChannelProperties.Maintenance := MaintenanceDemanded ChannelErrorType := “Dynamic frame packing function mismatch” ExtChannelErrorType := “Frame late error” (see Formula (65)) Alarm Notification.req(AREP, API, Alarm Priority, Alarm Type, Slot Number, Subslot Number, Alarm Specifier, Module Ident Number, Submodule Ident Number, Alarm Item)
Do nothing	This state change shall not indicate an event
[a] This value is used to illustrate the principle. The rules for ChannelProperties.Specifier apply.	

5.2.17.5 Coding of the field SubframeOffset

This field shall be coded as data type Unsigned16 with values according to Table 987.

Table 987 – SubframeOffset

Value (hexadecimal)	Meaning
0x0000 – 0x0005	Reserved
0x0006 – 0x059B	Number of octets from the begin of the frame C_SDU to the begin of the according subframe C_SDU
0x059C – 0xFFFF	Reserved

5.2.18 Coding section related to MRPD

5.2.18.1 Overview

The following rules shall be applied for the MediaRedundancyWatchDog of MRPD. The time value (for example 8 s) of the MediaRedundancyWatchDog shall be defined locally to avoid “false positive” events in case of frame errors. The MediaRedundancyWatchDog shall be activated when MRPD is used.

5.2.18.2 MediaRedundancyWatchDog expired

The checking whether the MediaRedundancyWatchDog (as an example with the time value 1 s) has expired shall be done according to Figure 191 and Table 988.

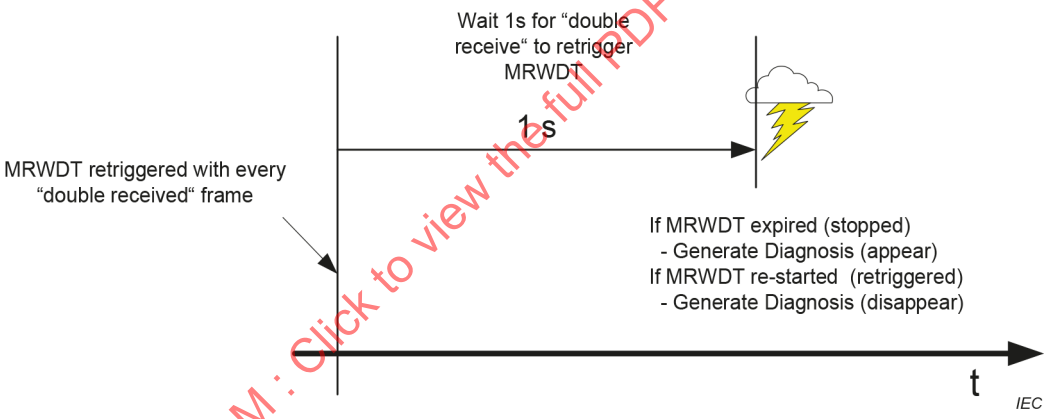


Figure 191 – MediaRedundancyWatchDog expired – appear and disappear

Table 988 – Event function table

Function name	Operations
Appear	Alarm Priority := ALARM_HIGH or ALARM_LOW Alarm Type:= MRPD problem notification User Structure Identifier := 0x8002 ChannelNumber := 0x8000 ChannelProperties.Specifier := Appears ChannelProperties.Maintenance :=MaintenanceDemanded ChannelErrorType := “Media redundancy with planned duplication mismatch” ExtChannelErrorType := “MRPD duplication void” (see Formula (65)) Alarm Notification.req(AREP, API, Alarm Priority, Alarm Type, Slot Number, Subslot Number, Alarm Specifier, Module Ident Number, Submodule Ident Number, Alarm Item)
Disappear	Alarm Priority := ALARM_HIGH or ALARM_LOW Alarm Type:= MRPD problem notification User Structure Identifier := 0x8002 ChannelNumber := 0x8000 ChannelProperties.Specifier := Disappears [a] ChannelProperties.Maintenance :=MaintenanceDemanded ChannelErrorType := “Media redundancy mismatch” ExtChannelErrorType := “MRPD duplication void” (see Formula (65)) Alarm Notification.req(AREP, API, Alarm Priority, Alarm Type, Slot Number, Subslot Number, Alarm Specifier, Module Ident Number, Submodule Ident Number, Alarm Item)
Do nothing	This state change shall not indicate an event
[a] This value is used to illustrate the principle. The rules for ChannelProperties.Specifier apply.	

5.2.19 Coding section related to controller to controller communication

5.2.19.1 Coding of the field FromOffsetData

This field shall be coded as data type Unsigned32 according to Table 989.

Table 989 – FromOffsetData

Value (hexadecimal)	Meaning	Description
0x00000000 – 0xFFFFFFFF	Mandatory	Contains the offset in octets as base offset for the request and the delivered chunk of data in the response.

5.2.19.2 Coding of the field NextOffsetData

This field shall be coded as data type Unsigned32 according to Table 990.

Table 990 – NextOffsetData

Value (hexadecimal)	Meaning	Description
0x00000000	Mandatory	The last chunk of data is read.
Other	Mandatory	Delivers the offset value to be used next for convenience. More data is available.

5.2.19.3 Coding of the field TotalSize

This field shall be coded as data type Unsigned32 according to Table 991.

Table 991 – TotalSize

Value (hexadecimal)	Meaning	Description
0x00000000	Reserved	—
Other	Mandatory	Contains the total size in octets of the requested object.

5.2.20 Coding section related to system redundancy

5.2.20.1 Coding of the field RedundancyInfo

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0: RedundancyInfo.EndPoint1
This field shall be coded according to Table 992, Figure 192, and Figure 193.

Table 992 – RedundancyInfo.EndPoint1

Value (hexadecimal)	Meaning
0x00	The mounting position of the delivering node in a rack is not the one to the left or the one above compared to EndPoint2.
0x01	The mounting position of the delivering node in a rack is the one to the left or the one above compared to EndPoint2.

Bit 1: RedundancyInfo.EndPoint2
This field shall be coded according to Table 993, Figure 192, and Figure 193.

Table 993 – RedundancyInfo.EndPoint2

Value (hexadecimal)	Meaning
0x00	The mounting position of the delivering node in a rack is not the one to the right or the one below compared to EndPoint1.
0x01	The mounting position of the delivering node in a rack is the one to the right or the one below compared to EndPoint1.

Valid combinations of RedundancyInfo.EndPoint1 and RedundancyInfo.EndPoint2 are shown in Table 994.

Table 994 – Valid combination of RedundancyInfo.EndPoint1 and RedundancyInfo.EndPoint2

RedundancyInfo. EndPoint1	RedundancyInfo. EndPoint2	Meaning
0x00	0x00	Reserved
0x00	0x01	Valid combination
0x01	0x00	Valid combination
0x01	0x01	Reserved

Figure 192 and Figure 193 show the naming scheme for EndPoint1 and Endpoint2.

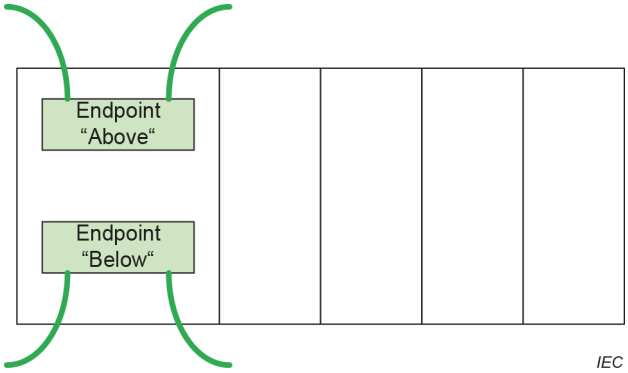


Figure 192 – EndPoint1 and Endpoint2 scheme – above and below

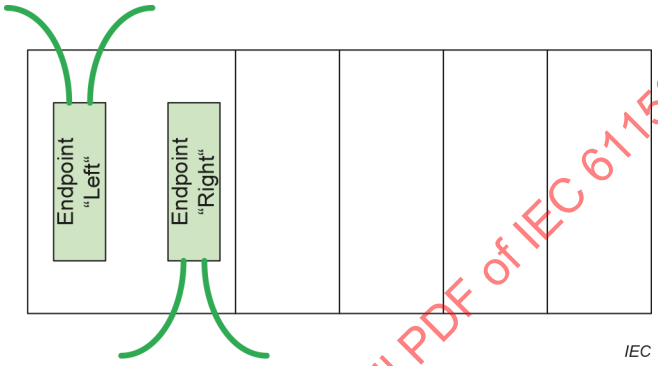


Figure 193 – EndPoint1 and Endpoint2 scheme – left and right

Bit 2 – 15: RedundancyInfo.Reserved
This field shall be set to zero.

5.2.20.2 Coding of the field SRProperties

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0: SRProperties.InputValidOnBackupAR
This field shall be coded according to Table 995 and Table 996.

Table 995 – SRProperties.InputValidOnBackupAR with SRProperties.Mode == 0

Value (hexadecimal)	Meaning
0x00	The IO controller shall not evaluate the input data.
0x01	The device shall deliver valid input data

Table 996 – SRProperties.InputValidOnBackupAR with SRProperties.Mode == 1

Value (hexadecimal)	Meaning
0x00	The IO device shall mark the data as invalid using APDU_Status.DataStatus.DataValid == Invalid.
0x01	The device shall deliver valid input data

Bit 1: SRProperties.Reserved_1
This field shall be coded according to Table 997.

Table 997 – SRProperties.Reserved_1

Value (hexadecimal)	Meaning
0x00	Shall be set to zero for this document.
0x01	Legacy mode

Bit 2: SRProperties.Mode
This field shall be coded according to Table 998.

Table 998 – SRProperties.Mode

Value (hexadecimal)	Meaning
0x00	The IO controller does not support APDU_Status.DataStatus.DataValid == Invalid if input data is marked as not valid.
0x01	Default The IO device shall use APDU_Status.DataStatus.DataValid == Invalid if input data is marked as not valid.

Bit 3 – 15: SRProperties.Reserved_2
This field shall be set to zero.

Bit 16 – 31: SRProperties.Reserved_3
This field shall be set according to 3.4.2.2.

5.2.20.3 Coding of the field RedundancyDataHoldFactor

This field shall be coded as data type Unsigned16 according to Table 999. The time base shall be one millisecond.

Table 999 – RedundancyDataHoldFactor

Value (hexadecimal)	Meaning	Description
0x0000 – 0x0002	Reserved	—
0x0003 – 0x00C7	Optional	An expiration of the time leads to an AR termination.
0x00C8 – 0xFFFF	Mandatory	An expiration of the time leads to an AR termination.

The RedundancyDataHoldTime is calculated according to Formula (85).

$$RDHT = RedundancyDataHoldFactor \times 1\ ms \quad (85)$$

where

RDHT is the redundancy data hold time

RedundancyDataHoldFactor is the factor for the calculation of the redundancy data hold time

5.2.20.4 Coding of the field NumberOfEntries

This field shall be coded as data type Unsigned16 according to Table 1000.

Table 1000 – NumberOfEntries

Value (hexadecimal)	Meaning	Use
0x0000 – 0xFFFF	Number of entries	—

5.2.21 Coding section related to energy saving

5.2.21.1 Coding of the field PE_OperationalMode

This field shall be coded as data type Unsigned8 and shall contain the values according to Table 1001.

Table 1001 – PE_OperationalMode

Value (hexadecimal)	Meaning	Comments
0x00	PE_PowerOff	—
0x01	PE_EnergySavingMode_1	Manufacturer specific mode_1
…	…	Manufacturer specific mode_x
0x1F	PE_EnergySavingMode_31	Manufacturer specific mode_31
0x20 – 0xEF	Reserved	—
0xF0	PE_Operate	—
0xF1 – 0xFD	Reserved	—
0xFE	PE_SleepModeWOL	—
0xFF	PE_ReadyToOperate	—

5.2.22 Coding section related to asset management

5.2.22.1 Coding section related to AM_Location

5.2.22.1.1 Coding of the field AM_Location with AM_Location.Structure == 0x01

The coding of this field shall be according to 3.4.2.3.7 and the individual bits shall have the following meaning:

Bit 0 – 7: AM_Location.Structure
This field shall be coded according to Table 1002.

Table 1002 – AM_Location.Structure

Value (hexadecimal)	Comment
0x00	Reserved
0x01	Twelve level tree format
0x02	Slot- and SubslotNumber format
0x03 – 0xFF	Reserved

Bit 8 – 17: AM_Location.Level0
This field shall be coded according to Table 1003.

The coding “Level not used” shall only be used, if all subsequent levels are not used, too.

Table 1003 – AM_Location.LevelIx

Value (hexadecimal)	Meaning
0x000 – 0x3FE	Address information to identify a reported node
0x3FF	Level not used

Bit 18 – 27: AM_Location.Level1
This field shall be coded according to Table 1003.

Bit 28 – 37: AM_Location.Level2
This field shall be coded according to Table 1003.

Bit 38 – 47: AM_Location.Level3
This field shall be coded according to Table 1003.

Bit 48 – 57: AM_Location.Level4
This field shall be coded according to Table 1003.

Bit 58 – 67: AM_Location.Level5
This field shall be coded according to Table 1003.

Bit 68 – 77: AM_Location.Level6
This field shall be coded according to Table 1003.

Bit 78 – 87: AM_Location.Level7
This field shall be coded according to Table 1003.

Bit 88 – 97: AM_Location.Level8
This field shall be coded according to Table 1003.

Bit 98 – 107: AM_Location.Level9
This field shall be coded according to Table 1003.

Bit 108 – 117: AM_Location.Level10
This field shall be coded according to Table 1003.

Bit 118 – 127: AM_Location.Level11
This field shall be coded according to Table 1003.

5.2.22.1.2 Coding of the field AM_Location with AM_Location.Structure == 0x02

The coding of this field shall be according to 3.4.2.3.7 and the individual bits shall have the following meaning:

Bit 0 – 7: AM_Location.Structure
This field shall be coded according to Table 1002.

Bit 8 – 15: AM_Location.Reserved1
This field shall be coded according to Table 1004.

Table 1004 – AM_Location.Reserved1

Value (hexadecimal)	Meaning
0x00	Reserved

Bit 16 – 31: AM_Location.BeginSlotNumber
This field shall be coded according to 5.2.3.2 with the SlotNumber at which the reported asset starts.

Bit 32 – 47: AM_Location.BeginSubslotNumber
This field shall be coded according to Table 1005 with the SubslotNumber at which the reported asset starts.

Table 1005 – AM_Location.BeginSubslotNumber

Value (hexadecimal)	Meaning
0x0000 – 0xFFFE	See 5.2.3.3
0xFFFF	Asset covers whole slot

Bit 48 – 63: AM_Location.EndSlotNumber
This field shall be coded according to 5.2.3.2 with the SlotNumber at which the reported asset ends.

Bit 64 – 79: AM_Location.EndSubslotNumber
This field shall be coded according to Table 1006 with the SubslotNumber at which the reported asset ends.

Table 1006 – AM_Location.EndSubslotNumber

Value (hexadecimal)	Meaning
0x0000 – 0xFFFE	See 5.2.3.3
0xFFFF	Asset covers whole slot

Bit 80 – 95: AM_Location.Reserved2
This field shall be coded according to Table 1007.

Table 1007 – AM_Location.Reserved2

Value (hexadecimal)	Meaning
0x0000	Reserved

Bit 96 – 111: AM_Location.Reserved3
This field shall be coded according to Table 1008.

Table 1008 – AM_Location.Reserved3

Value (hexadecimal)	Meaning
0x0000	Reserved

Bit 112 – 127: AM_Location.Reserved4
This field shall be coded according to Table 1009.

Table 1009 – AM_Location.Reserved4

Value (hexadecimal)	Meaning
0x0000	Reserved

5.2.22.2 Coding section related to asset management string

5.2.22.2.1 General

All data with type UnicodeString8 shall be left justified. If the text is shorter than the defined string length, the gap shall be filled with blanks.

5.2.22.2.2 Coding of the field AM_SoftwareRevision

This field shall be coded as data type UnicodeString8[64] and contains the manufacturer defined software revision. The support of IM_Software_Revision is preferred.

If the field IM_Software_Revision is supported, then this field shall be filled with blanks.

If the field AM_SoftwareRevision is supported, then IM_Software_Revision shall be set according to 4.10.3.3.4.2.

5.2.22.2.3 Coding of the field AM_HardwareRevision

This field shall be coded as data type UnicodeString8[64] and contains the manufacturer defined hardware revision. The support of IM_Hardware_Revision is preferred.

If the field IM_Hardware_Revision is supported, then this field shall be filled with blanks.

If the field AM_HardwareRevision is supported, then IM_Hardware_Revision shall be set to zero.

5.2.22.2.4 Coding of the field AM_DeviceIdentification

The coding of this field shall be according to 3.4.2.3.6 and the individual bits shall have the following meaning:

Bit 0 – 15: AM_DeviceIdentification.DeviceSubID
This field shall be coded according to Table 1010 and Table 1011. The definition is dependent on the AM_DeviceIdentification.DeviceID.

Table 1010 – AM_DeviceIdentification.DeviceSubID

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Usage defined by AM_DeviceIdentification.Organization

Table 1011 – AM_DeviceIdentification.DeviceSubID for AM_DeviceIdentification.Organization := 0x0000

Value (hexadecimal)	Meaning
0x0000	For this document
0x0001 – 0xFFFF	Reserved

Bit 16 – 31: AM_DeviceIdentification.DeviceID
This field shall be coded according to Table 1012. The definition is dependent on the AM_DeviceIdentification.VendorID.

Table 1012 – AM_DeviceIdentification.DeviceID

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Contains the DeviceID

Bit 32 – 47: AM_DeviceIdentification.VendorID
This field shall be coded according to Table 1013. The definition is dependent on the AM_DeviceIdentification.Organization.

Table 1013 – AM_DeviceIdentification.VendorID

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Contains the VendorID

Bit 48 – 63: AM_DeviceIdentification.Organization
This field shall be coded according to Table 1014.

Table 1014 – AM_DeviceIdentification.Organization

Value (hexadecimal)	Meaning
0x0000	For this document
0x0001 – 0x00FF	Reserved
0x0100	IEC 61131-9
0x0101	IEC 61158 Type 3
0x0102	IEC 61158 Type 6
0x0103	IEC 62026-2
0x0104	IEC 61158 Type 9
0x0105	IEC 61158 Type 2
0x0106	IEC 61158 Type 1
0x0107	EN 50325-4
0x0108	IEC 61158 Type 8
0x0109 – 0xFFFF	Reserved

5.2.22.3 Coding of the field AM_TypeIdentification

This field shall be coded as data type Unsigned16 and contains the manufacturer assigned type identification according to Table 701.

Any manufacturer specific values shall be defined according to the AM_DeviceIdentification.

5.2.22.4 Coding of the field AM_Reserved

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 15: AM_Reserved.Reserved
This field shall be set to zero.

5.2.23 Coding section related to reporting system

5.2.23.1 Coding of the field RS_AlarmInfo

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: RS_AlarmInfo.Reserved1
This field shall be set to zero.

Bit 8 – 15: RS_AlarmInfo.Reserved2
This field shall be set according to 3.4.2.2.

5.2.23.2 Coding of the field RS_Properties

The coding of this field shall be according to 3.4.2.3.5 and the individual bits shall have the following meaning:

Bit 0: RS_Properties.AlarmTransport
This field shall be set according to Table 1015.

Table 1015 – RS_Properties.AlarmTransport

Value (hexadecimal)	Meaning
0x0	Default Reporting system events need to be read and acknowledged by defined record service
0x1	Optional Reporting system events shall be forwarded to the IO controller using the Alarm transport Reporting system events may be read and acknowledged by defined record service

Bit 1 – 23: RS_Properties.Reserved1
This field shall be set to zero.

Bit 24 – 31: RS_Properties.Reserved2
This field shall be set according to 3.4.2.2.

5.2.23.3 Coding of the field RS_BlockType

This field shall be coded as data type Unsigned16 with the values according to Table 1016 and Table 1017.

Table 1016 – RS_BlockType used for events

Value (hexadecimal)	Name	Comment
0x0000	Reserved	—
0x0001 – 0x3FFF	Manufacturer specific	—
0x4000	RSO_StatusObserver	Observer status observer Checks whether a given observer is running normally. If not, an event will be posted.
0x4001	RSO_BufferObserver	Buffer status observer Checks whether the RS event buffer is running normally. If for example an overflow occurs, an event will be posted.
0x4002	RSO_TimeStatus	Time status observer Checks whether the time source is running normally. If for example synchronization is lost, an event will be posted.
0x4003	RSO_SRLObserver	System redundancy layer observer Checks whether a switchover occurs. If so, events will be posted showing begin and end of the switchover.
0x4004	RSO_SourceIdentification	Source identification observer Posts identity information events.
0x4005 – 0x400F	Reserved	—
0x4010	RSO_DigitalInputObserver	Digital input observer Checks whether the status of an input changes and for example will post an event for each change.
0x4011 – 0x6FFF	Reserved for normative usage	—
0x7000 – 0x7FFF	Reserved for profile usage	—

Table 1017 – RS_BlockType used for adjust

Value (hexadecimal)	Comment
0x8000	Reserved
0x8001 – 0xBFFF	Manufacturer specific
0xC000 – 0xC00F	Reserved for normative usage
0xC010	Digital input observer RSO_DigitalInputObserver
0xC011 – 0xEFFF	Reserved for normative usage
0xF000 – 0xFFFF	Reserved for profile usage

5.2.23.4 Coding of the field RS_BlockLength

This field shall be coded as data type Unsigned16 with the values according to Table 1018 and Table 1019.

Table 1018 – RS_BlockLength in conjunction with RS_EventBlock

Value (hexadecimal)	Meaning
0x0000 – 0x0004	Reserved
0x0005 – 0x0084	Number of octets without counting the fields RS_BlockType and RS_BlockLength
0x0085 – 0xFFFF	Reserved

Table 1019 – RS_BlockLength in conjunction with other blocks

Value (hexadecimal)	Meaning
0x0000 – 0x0002	Reserved
0x0003 – 0xFFFF	Number of octets without counting the fields RS_BlockType and RS_BlockLength

5.2.23.5 Coding of the field RS_Specifier

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 10: RS_Specifier.SequenceNumber
This field shall be set according to Table 1020.

By means of the RS_Specifier.SequenceNumber the sink detects duplications and the source detects which events are acknowledged. The sequence number should start with zero and shall be incremented by 1 with every event added to the RS ASE. The receiver shall accept an arbitrary value as start value.

Each AR or ARset administrates its own RS_Specifier.SequenceNumber.

Table 1020 – RS_Specifier.SequenceNumber

Value (hexadecimal)	Comment
0x0000 – 0x07FF	Sequence number of a stored event

Bit 11 – 13: RS_Specifier.Reserved
This field shall be set to zero.

Bit 14 – 15: RS_Specifier.Specifier
This field shall be set according to Table 1021.

Table 1021 – RS_Specifier.Specifier

Value (hexadecimal)	Meaning
0x0	Current value
0x1	Appears
0x2	Disappears
0x3	Reserved

5.2.23.6 Coding of the field RS_MinusError

This field shall be coded as data type Unsigned16 with the values according to Table 1022. It contains the maximum value of the event capture error (assumed capture time too early) defined by the device implementation in relation to the local maintained time (TIME ASE).

Table 1022 – RS_MinusError

Value (hexadecimal)	Comment
0x0000 – 0x2710	Maximum negative deviation of the Time_TimeStamp in units of 10 µs
0x2711 – 0xFFFF	Reserved

5.2.23.7 Coding of the field RS_PlusError

This field shall be coded as data type Unsigned16 with the values according to Table 1023. It contains the maximum value of the event capture error (assumed capture time too late) defined by the device implementation in relation to the local maintained time (TIME ASE).

Table 1023 – RS_PlusError

Value (hexadecimal)	Comment
0x0000 – 0x2710	Maximum positive deviation of the Time_TimeStamp in units of 10 µs
0x2711 – 0xFFFF	Reserved

5.2.23.8 Coding of the field RS_ExtensionBlockType

This field shall be coded as data type Unsigned8 with the values according to Table 1024. Each RS_BlockType creates its own RS_ExtensionBlockType namespace.

Table 1024 – RS_ExtensionBlockType

Value (hexadecimal)	Comment
0x00	Reserved
0x01 – 0x3F	Manufacturer specific usage
0x40 – 0x7F	Reserved
0x80 – 0xBF	Reserved for normative usage
0xC0 – 0xFF	Reserved for profile usage

5.2.23.9 Coding of the field RS_ExtensionBlockLength

This field shall be coded as data type Unsigned8 with the values according to Table 1025.

Table 1025 – RS_ExtensionBlockLength

Value (hexadecimal)	Comment
0x00 – 0x01	Reserved
0x02 – 0x2F	Length of Data
0x30 – 0xFF	Reserved

5.2.23.10 Coding of the field RS_MaxScanDelay

This field shall be coded as data type Unsigned16 with the values according to Table 1026.

Table 1026 – RS_MaxScanDelay

Value (hexadecimal)	Comment
0x0000	Reserved
0x0001 – 0x2710	Maximum scan delay for the supervised channel in 10 µs.
0x2711 – 0xFFFF	Reserved

5.2.23.11 Coding of the field RS_AdjustSpecifier

The coding of this field shall be according to 3.4.2.3.3 and the individual bits shall have the following meaning:

Bit 0 – 5: RS_AdjustSpecifier.Reserved
This field shall be set to zero.

Bit 6 – 7: RS_AdjustSpecifier.Incident
This field shall be set according to Table 1027.

Table 1027 – RS_AdjustSpecifier.Incident

Value (hexadecimal)	Comment
0x00	Reserved
0x01	Rising edge
0x02	Falling edge
0x03	Reserved

5.2.23.12 Coding of the field RS_ReasonCode

The coding of this field shall be according to 3.4.2.3.5. The individual bits shall have the following meaning:

Bit 0 – 15: RS_ReasonCode.Reason
This field shall be set according to Table 1028.

Table 1028 – RS_ReasonCode.Reason

Value (hexadecimal)	Comment
0x0000	Reserved
0x0001	Observed data status unclear
0x0002	Buffer overflow
0x0003 – 0xFFFF	Reserved

Bit 16 – 31: RS_ReasonCode.Detail
This field shall be set according to Table 1029.

Table 1029 – RS_ReasonCode.Detail

Value (hexadecimal)	Comment
0x0000	No detail
0x0001 – 0xFFFF	Reserved

5.2.23.13 Coding of the field RS_DigitalInputCurrentValue

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0: RS_DigitalInputCurrentValue.Value
This field shall be set according to Table 1030.

Table 1030 – RS_DigitalInputCurrentValue.Value

Value (hexadecimal)	Comment
0x00	Digital input is zero
0x01	Digital input is one

Bit 1 – 15: RS_DigitalInputCurrentValue.Reserved
This field shall be set to zero.

5.2.23.14 Coding of the field RS_DomainIdentification

This field shall be coded as data type OctetString[16] with the values according to Table 1031.

Table 1031 – RS_DomainIdentification

Value (hexadecimal)	Comment
zero	Unknown The reporting system uses the local (arbitrary timescale) time or No domain known
Other	Domain identification, for example, a Domain UUID

5.2.23.15 Coding of the field RS_MasterIdentification

This field shall be coded as data type OctetString[8] with the values according to Table 1032.

Table 1032 – RS_MasterIdentification

Value (hexadecimal)	Comment
zero	Unknown, the reporting system uses the local (arbitrary timescale) time
Other	Master identification, for example, the MAC address of the synchronization master in EUI-64 format according to IEEE Std 802

The RS_DomainIdentification and RS_MasterIdentification show the state at the time the event was detected.

5.2.24 Coding section related to logbook

5.2.24.1 Coding of the field ActualLocalTimeStamp

This field shall be coded as data type Unsigned64 according to Table 1033.

Table 1033 – ActualLocalTimeStamp

Value (hexadecimal)	Meaning	Use
0x0000000000000000 – 0xFFFFFFFFFFFFFFFF	Contains the current cycle count value when reading the logbook.	—

5.2.24.2 Coding of the field LocalTimeStamp

This field shall be coded as data type Unsigned64 according to Table 1034.

Table 1034 – LocalTimeStamp

Value (hexadecimal)	Meaning	Use
0x0000000000000000 – 0xFFFFFFFFFFFFFFFF	Contains the current cycle count when storing the entry to the logbook	—

5.2.24.3 Coding of the field NumberOfLogEntries

This field shall be coded as data type Unsigned16 according to Table 1035.

Table 1035 – NumberOfLogEntries

Value	Meaning	Use
0	Reserved	—
Other	Number of log entries	—

5.2.24.4 Coding of the field EntryDetail

This field shall be coded as data type Unsigned32 according to protocol machine behavior and Table 1036.

Table 1036 – EntryDetail

Value (hexadecimal)	Meaning	Use
0x00000000	No detail	—
Other	Value derived from the signaling protocol machine	—

5.2.25 Coding section related to Time

5.2.25.1 Coding of the field Time_TimeStamp

This field shall be coded as data type TimeStamp (see 4.1.2.8) with the values according to Table 1037.

Table 1037 – Time_TimeStamp

Value (hexadecimal)	Comment
Other	Point in time fetched from TIME ASE when the event occurs.

5.2.26 Coding section related to Channel Related Process Alarm Reason

5.2.26.1 Overview

The following coding section applies to the Channel Related Process Alarm Reason functionality. Allowed combinations of the defined fields are shown in Table 1038.

Table 1038 – Allowed combinations of PRAL_Reason, PRAL_ExtReason, and PRAL_ReasonAddValue

PRAL_Reason	PRAL_ExtReason	PRAL_ReasonAddValue	Usage
Normative	Normative	Normative	Allowed
Normative	Vendor specific	Vendor specific	Allowed
Normative	Profile specific	Profile specific	Allowed
Profile specific	Profile specific	Profile specific	Allowed
Profile specific	Vendor specific	Vendor specific	Allowed
Profile specific	Normative	Normative	Prohibited
Vendor specific	Vendor specific	Vendor specific	Allowed
Vendor specific	Normative	Normative	Prohibited
Vendor specific	Profile specific	Profile specific	Prohibited

5.2.26.2 Coding of the field PRAL_ChannelProperties

The coding of this field shall be according to 3.4.2.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: PRAL_ChannelProperties.Reserved_1
This field shall be set according to Table 1039.

Table 1039 – PRAL_ChannelProperties.Reserved_1

Value (hexadecimal)	Meaning	Usage
0x00 – 0xFF	Reserved	—

Bit 8: PRAL_ChannelProperties.Accumulative
This field shall be set according to Table 1040.

Table 1040 – PRAL_ChannelProperties.Accumulative

Value (hexadecimal)	Meaning	Usage
0x00	Single	Single channel Reason for the reported channel only
0x01	Accumulative	Multiple channel Accumulative reason for more than one channel

Bit 9 – 12: PRAL_ChannelProperties.Reserved_2
This field shall be set according to Table 1041.

Table 1041 – PRAL_ChannelProperties.Reserved_2

Value (hexadecimal)	Meaning	Usage
0x00 – 0x0F	Reserved	—

Bit 13 – 15: PRAL_ChannelProperties.Direction
This field shall be set according to Table 1042.

Table 1042 – PRAL_ChannelProperties.Direction

Value (hexadecimal)	Meaning
0x00	Manufacturer specific
0x01	Input
0x02	Output
0x03	Input/Output
0x04 – 0x07	Reserved

5.2.26.3 Coding of the field PRAL_Reason

This field shall be coded as data type Unsigned16 with the values according to Table 1043.

Table 1043 – Values for PRAL_Reason

Value (hexadecimal)	Meaning	Assigned text
0x0000	Reserved	Unknown reason
0x0001	Internal gate opening (Gate start)	Counter or way measurement starts Internal gate opening (Gate start)
0x0002	Internal gate closing (Gate stop)	Counter or way measurement stops Internal gate closing (Gate stop)
0x0003	Overflow (High counting limit violated)	Counter or way measurement register Overflow (High counting limit violated)
0x0004	Underflow (Low counting limit violated)	Counter or way measurement register Underflow (Low counting limit violated)

Value (hexadecimal)	Meaning	Assigned text
0x0005	Compare event for DQ0 has occurred	Compare of counter or way measurement register with a parameterized value “DQ0” (or “first digital output”) shows hit (Compare event for “first digital output” has occurred)
0x0006	Compare event for DQ1 has occurred	Compare of counter or way measurement register with a parameterized value “DQ1” (or “second digital output”) shows hit (Compare event for “second digital output” has occurred)
0x0007	Zero pass	Counter or way measurement signed register hits or passes zero (Zero pass)
0x0008	New capture value available	New capture value available
0x0009	Synchronization of the counter by an external signal	Counter or way measurement register value is set by an external signal (Synchronization of the counter by an external signal)
0x000A	Direction reversal	Counter or way measurement register changes counting direction (Direction reversal)
0x000B – 0x000F	Reserved	Unknown reason
0x0010	Rising edge	The digital channel detects a rising edge (Rising edge)
0x0012	Falling edge	The digital channel detects a falling edge (Falling edge)
0x0013	Upper user limit 1 exceeded	The parameterized compare value 1 for the channel is exceeded (Upper user limit 1 exceeded)
0x0014	Lower user limit 1 undershot	The parameterized compare value 1 for the channel is undershot (Lower user limit 1 undershot)
0x0015	Upper user limit 2 exceeded	The parameterized compare value 2 for the channel is exceeded (Upper user limit 2 exceeded)
0x0016	Lower user limit 2 undershot	The parameterized compare value 2 for the channel is undershot (Lower user limit 2 undershot)
0x0017	Upper user limit 3 exceeded	The parameterized compare value 3 for the channel is exceeded (Upper user limit 3 exceeded)
0x0018	Lower user limit 3 undershot	The parameterized compare value 3 for the channel is undershot (Lower user limit 3 undershot)
0x0019	Upper user limit 4 exceeded	The parameterized compare value 4 for the channel is exceeded Upper user limit 4 exceeded
0x001A	Lower user limit 4 undershot	The parameterized compare value 4 for the channel is undershot (Lower user limit 4 undershot)
0x001B – 0x00FF	Reserved	Unknown reason
0x0100 – 0x7FFF	Manufacturer specific	Manufacturer specific reason
0x8000 – 0x8FFF	Reserved	Reserved
0x9000 – 0x9FFF	Reserved for profiles	Profile specific reason
0xA000 – 0xFFFF	Reserved	Reserved

5.2.26.4 Coding of the field PRAL_ExtReason

This field shall be coded as data type Unsigned16 with the values according to Table 1044. The value of this field depends on the field PRAL_Reason.

Table 1044 – Values for PRAL_ExtReason

Value (hexadecimal)	Meaning	Assigned text
0x0000	Unspecified	Unspecified
0x0001 – 0x7FFF	Manufacturer specific	Manufacturer specific reason detail
0x8000 – 0x8FFF	Reserved	Reserved
0x9000 – 0x9FFF	Reserved for profiles	Profile specific reason detail
0xA000 – 0xFFFF	Reserved	Reserved

5.2.26.5 Coding of the field PRAL_ReasonAddValue

5.2.26.5.1 General

This field shall be coded as data type OctetString[] with 4 to 128 octets according to Table 1045, Table 1046, Table 1047, and Table 1222.

Table 1045 – Usage of PRAL_ReasonAddValue

Length (hexadecimal)	Meaning
0x04	Definition from Table 1046 applies
0x05 – 0x80	Definition from Table 1047 applies
Other	Reserved

Table 1046 – Values for PRAL_ReasonAddValue[0..3]

Value (hexadecimal)	Meaning
0x00, 0x00, 0x00, 0x00	No additional information – independent of PRAL_Reason and PRAL_ExtReason
Other	Definition depending on PRAL_Reason and PRAL_ExtReason

Table 1047 – Values for PRAL_ReasonAddValue[0] to [127]

Value (hexadecimal)	Meaning
0x00 – 0xFF	Definition depending on PRAL_Reason and PRAL_ExtReason

5.2.27 Void

Intentionally left blank.

5.3 FAL protocol state machines

5.3.1 Overall structure

5.3.1.1 Overview

The FAL protocol state machine structure is as defined in Figure 194. The general structure is according to the IEC 61158-6 series protocol machine model.

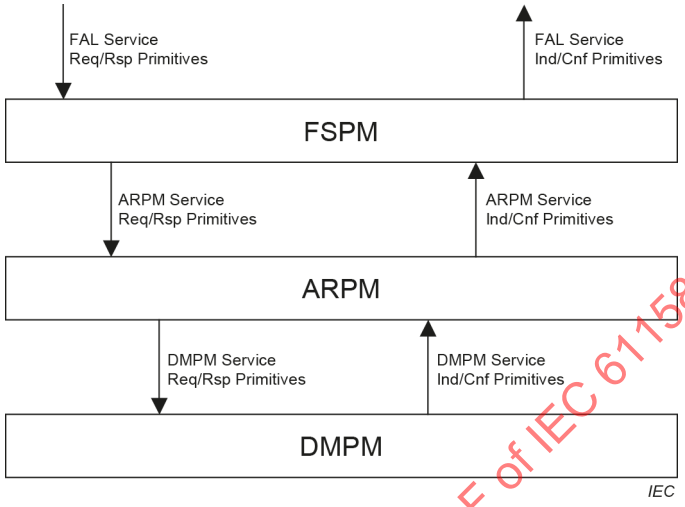


Figure 194 – Relationship among Protocol Machines

The behavior of the FAL is specified by three integrated protocol machines. The FSPM is the service interface between the FAL services that are part of the FAL Class specification and the particular AREP.

The FAL provides two kinds of protocol machine architectures, one for IO controller and one for IO devices.

The FSPM is responsible for the following activities:

– To accept service primitives from the FAL service user and convert them into FAL internal primitives.
– To select the ARPM state machine based on the implicit addressing mechanism and send FAL internal primitives with the service parameters to the ARPM.
– To accept FAL internal primitives from the ARPM and convert them into service primitives for the FAL service user.
– To deliver the FAL service primitives to the FAL user.

The ARPM specifies the conveyance type for the application relation.

The DMPM specifies the mapping to the Data Link Layer. The DMPM therefore defines two protocol machines, the LMPM and the MAC protocol machines.

5.3.1.2 FAL Service Protocol Machine

The FSPM State Machine coordinates the underlying state machines used for processing of the various services and application relations.

The FSPM basically is a mapping protocol machine. The main task is to pass the service to the protocol machine responsible for that service and forward confirmations and responses to the user.

5.3.1.3 IO controller to IO device

The state machines, described in 5.6, are responsible for the cyclic, acyclic and alarm data transfer between an IO controller and an IO device. These are used to establish an IO AR.

5.3.1.4 IO supervisor to IO device

The state machines, described in 5.6, are responsible for the cyclic, acyclic and alarm data transfer between an IO supervisor and an IO device. These are used to establish a Supervisor AR.

5.3.1.5 Network management entity to IO controller and to IO device

If a time-aware system is configured, the state machines NME, BNME, NCE, TDE, PCE and NUE are responsible to establish a network configuration, useable by the IO AR and the Supervisor AR.

5.3.1.6 DLL Mapping Protocol Machines

The DL Mapping Protocol Machine (DMPM) connects the other State machines and Layer 2. DMPM provides the coordination of all state machines concerning the configuration and error handling of the Data Link Layer Usage. The functions are mapped by the DMPM to the DLL services of Layer 2. The DMPM generates the necessary Layer 2 parameters of the service, receives the confirmations and indications from Layer 2 and passes them to the appropriate DMPM-User.

The DMPM is implemented directly at the DLL User – DLL interface as described in Data Link Layer Service Definitions (refer to Enhanced Internal Sublayer Services IEEE Std 802.3).

For the mapping of the DMPM functions to Layer 2, the DMPM uses the following services:

– MA_UNITDATA.req
– MA_UNITDATA.ind

For the purpose of this specification, there is an explicit confirmation for the MA_UNITDATA request service primitive indicating that the DLPDU has been transmitted.

5.4 AP-Context state machine

There is no AP-Context state machine defined for this Protocol.

5.5 FAL Service Protocol Machines

5.5.1 Overview

This type specifies one FAL Service Protocol Machine (FSPM) state machine to transfer the FAL user service primitives into FAL internal service primitives and vice versa. There are four kinds of FSPM defined, one for all kind of entities (FSPMPON), one for the IO device entity (FSPMDEV), one for the IO controller entity (FSPMCTL), and one for the Network Management entity (FSPMNME).

5.5.2 FAL Service Protocol Machine Power-On

5.5.2.1 Overview

The FSPMPON is responsible for the start of the underlying components:

– CIM,
– Device,

– Controller, and

– Network Management Entity.

CIM starts the protocol machines LMPM, LLDP, DHCP, DCP, IP, UDP, RPC, DCP, PTCP, ALPM, PPM, CPM..., Device starts the protocol machine CMDEV, Controller starts the protocol machine CMCTL, and Network Management Entity starts the protocol machine NME.

5.5.2.2 Primitive definitions

Table 1048 shows the service primitives including their associated parameters issued by the AP-Context (FAL user) and received by the FSPMPON with the mapping to the underlying services.

Table 1048 – Primitives issued by AP-Context (FAL user) to FSPMPON

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
–	–	–	–

Table 1049 shows the service primitives including their associated parameters issued by the FSPMPON and received by the AP-Context (FAL user).

Table 1049 – Primitives issued by FSPMPON to AP-Context (FAL user)

Primitive mapped from	Parameter mapped from	Primitive name	Associated parameters
–	–	–	–

5.5.2.3 State transition diagram

The FSPMPON defines no state machine. Therefore, no states are described.

5.5.2.4 State machine description

The FSPMPON defines no state machine. Therefore, no state machine description exists.

5.5.2.5 FSPMPON state table

The FSPMPON defines no state machine because all services are transferred to the underlying protocol machines.

5.5.2.6 Functions, Macros, Timers and Variables

The FSPMPON defines no functions, macros, timers and variables. Therefore, no description exists.

5.5.3 FAL Service Protocol Machine Device

5.5.3.1 Overview

The FSPMDEV provides an interface for the user of the Device component.

5.5.3.2 Primitive definitions

Table 1050 shows the service primitives including their associated parameters issued by the AP-Context (FAL user) and received by the FSPMDEV with the mapping to the underlying services.

Table 1050 – Primitives issued by AP-Context (FAL user) to FSPMDEV

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
local AR Abort.req	AREP	CM_Abort_req (AREP)	AREP=AREP
local Create LogBook Entry	info	CreateLogBookEntry (info)	info = info
local Get Input IOCS.req	AREP CREP Slot Number Subslot Number	CPM_Get_Data_req (CREP)	MAP_GINIOCS_REQ
local Get Output.req	AREP CREP Slot Number Subslot Number	CPM_Get_Data_req (CREP), CPM_Get_Status_req (CREP)	CREP=CREP
local Set AR State.req	AREP ARstate	Set_Variable (ARstate)	AREP=AREP ARstate=ARstate
local Set Command.rsp (+)	List Of Response	DCP_Set.rsp (+)	CREP DA ListOfResponse = List Of Response
local Set Command.rsp (-)	ERRCLS ERRCODE	DCP_Set.rsp (-)	CREP DA ERRCLS = ERRCLS ERRCODE = ERRCODE
local Set Input.req	AREP CREP Slot Number Subslot Number IOPS Subslot Input Data	PPM_Set_Data_req (CREP, Data)	MAP_SIN_REQ
local Set Output IOCS.req	AREP CREP Slot Number Subslot Number IOCS	PPM_Set_Data_req (CREP, Data)	MAP_OIOCS_REQ
local Set Provider State.req	AREP ProviderState Flag	PPM_Set_Status_req (CREP, D_Status)	MAP_SPS_REQ
local Set Redundancy.req	AREP Redundancy Flag	PPM_Set_Status_req (CREP, D_Status)	MAP_SRS_REQ
local Set State.req	AREP State Flag	PPM_Set_Status_req (CREP, D_Status)	MAP_SSS_REQ
Application Ready.req	AREP Session Key Alarm Sequence Number Module Diff Block	CM_Ccontrol.req (AREP, ControlBlock, ModuleDiffBlock)	MAP_AREADY_REQ
Connect.rsp (-)	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Connect.rsp (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_CON_RSP-

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
Connect.rsp (+)	AREP AR Response Block List of IO CR Response Blocks Alarm CR Response Block Module Diff Block AR RPC Block AR Vendor Block	CM_Connect.rsp (+)(AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock, ARRPCBlock, ARVendorBlock)	MAP_CON_RSP+
Prm Begin.rsp (-)	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Dcontrol.rsp (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_BOP_RSP-
Prm Begin.rsp (+)	AREP Session Key	CM_Dcontrol.rsp (+) (AREP, ControlBlock)	MAP_BOP_RSP+
Prm End.rsp (-)	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Dcontrol.rsp (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_EOP_RSP-
Prm End.rsp (+)	AREP Session Key Alarm Sequence Number	CM_Dcontrol.rsp (+) (AREP, ControlBlock)	MAP_EOP_RSP+
Read.rsp (-)	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_RSP-
Read.rsp (+)	AREP Seq Number Length Data	CM_Read.rsp (+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RSP+
Write.rsp (-)	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp (-) (AREP, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_RSP-
Write.rsp (+)	AREP Seq Number	CM_Write.rsp (+) (AREP, Seq Number, AddData1, AddData2)	MAP_WRITE_RSP+

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
Alarm.req	AREP API Alarm Priority Alarm Type Slot Number Subslot Number Alarm Specifier Module Ident Number Submodule Ident Number Alarm Item	ALPMI_Alarm_Notification.req (CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, Module_Ident_Number, Submodule_Ident_Number, Alarm_User_Data_Structure_Identifier, Alarm_User_Data)	MAP_AN_REQ
Alarm.rsp (-)	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier PNIO Status	ALPMR_Alarm_Ack.req (CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, PNIO_Status)	MAP_ALARM_RSP-
Alarm.rsp (+)	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier PNIO Status	ALPMR_Alarm_Ack.req (CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, PNIO_Status)	MAP_ALARM_RSP+

Table 1051 shows the service primitives including their associated parameters issued by the FSPMDEV and received by the AP-Context (FAL user).

Table 1051 – Primitives issued by FSPMDEV to AP-Context (FAL user)

Primitive mapped from	Parameter mapped from	Primitive name	Associated parameters
CM_Abort_cnf (-)	AREP=AREP	local AR Abort.cnf (-)	AREP
CM_Abort_cnf (+)	AREP=AREP	local AR Abort.cnf (+)	AREP
CMDEV_state_ind (state, SessionKey)	if (state == ABORT) then AREP=AREP Session Key=SessionKey else ignore	local AR Abort.ind	AREP, Session Key
CM_Release.ind (AREP, ReleaseBlock)	MAP_REL_IND	local AR Abort.ind	AREP Session Key
CMDEV_state_ind (state)	if (state == DATA) then AREP=AREP else ignore	local AR In Data.ind	AREP
CPM_NewData_ind (AREP, CREP, APDU_Status)	MAP_NEWSTATUS_IND	local Data State Changed.ind	AREP CREP DataValid Flag State Flag Redundancy Flag ProviderState Flag ProblemIndicator Flag

Primitive mapped from	Parameter mapped from	Primitive name	Associated parameters
DiagnosisEvent (AREP, CREP, API, Diagnosis Data)	MAP_DIAGEVT_IND	local Diagnosis Event	AREP CREP Alarm Item
CPM_Get_Status_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_GINIOCS_CNF-	local Get Input IOCS.cnf (-)	AREP
CPM_Get_Status_cnf (+) (CREP, Status, RecvCounter)	MAP_GINIOCS_CNF+	local Get Input IOCS.cnf (+)	AREP IOCS
CPM_Get_Data_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_GOUTD_CNF-	local Get Output.cnf (-)	AREP
CPM_Get_Data_cnf (+) (CREP, Data, New_Flag)	MAP_GOUTD_CNF+	local Get Output.cnf (+)	AREP IOPS Subslot Output Data New Flag IOCS
CPM_NewData_ind (AREP, CREP, APDU_Status, data)	MAP_NEWDATA_IND	local New Output.ind	AREP CREP Slot Number Subslot Number InData Flag
if (Set_Variable (ARstate)) == FALSE	AREP=AREP	local Set AR State.cnf (-)	AREP
if (Set_Variable (ARstate)) == TRUE	AREP=AREP	local Set AR State.cnf (+)	AREP
DCP_Set.ind	CREP, ListOfData, ListOfControlCommands	local Set Command.ind	DA List Of Data = ListOfData List Of Control Commands = ListOfControlCommands
PPM_Set_Data_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_SIN_CNF	local Set Input.cnf (-)	AREP
PPM_Set_Data_cnf (+) (CREP)	MAP_SIN_CNF	local Set Input.cnf (+)	AREP
PPM_Set_Data_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_OIOCS_CNF	local Set Output IOCS.cnf (-)	AREP
PPM_Set_Data_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_OIOCS_CNF	local Set Output IOCS.cnf (+)	AREP
PPM_Set_Status_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_SPS_CNF	local Set Provider State.cnf (-)	AREP
PPM_Set_Status_cnf (+) (CREP)	MAP_SPS_CNF	local Set Provider State.cnf (+)	AREP
PPM_Set_Status_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_SPS_CNF	local Set Redundancy.cnf (-)	AREP

Primitive mapped from	Parameter mapped from	Primitive name	Associated parameters
PPM_Set_Status_cnf (+) (CREP)	MAP_SPS_CNF	local Set Redundancy.cnf (+)	AREP
PPM_Set_Status_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_SPS_CNF	local Set State.cnf (-)	AREP
PPM_Set_Status_cnf (+) (CREP)	MAP_SPS_CNF	local Set State.cnf (+)	AREP
SyncStateChange_ind (Status)	MAP_SYSTI_IND	local Sync State Info.ind	Sync Data Sync Error Status
TickEvent (CycleCounter, CycleOffset)	MAP_SYNCH_IND	local SYNCH Event.ind	Slot Subslot Global Cycle Counter Phase Status
CM_Ccontrol.cnf (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_AREADY_CNF-	Application Ready.cnf (-)	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2
CM_Ccontrol.cnf (+)(AREP, ControlBlock)	MAP_AREADY_CNF+	Application Ready.cnf (+)	AREP Session Key Alarm Sequence Number
CM_Connect.ind (AREP ARBlockReq ListOfIOCRBlockReq ListOfExpectedSubmoduleBlockReq AlarmCRBlockReq ParameterServerBlock ListOfMulticastCRBlock ARRPCBlock IRInfosBlock SRInfoBlock ARVendorBlock ARServerBlock ARFSUBlock)	MAP_CON_IND	Connect.ind	AREP AR Parameter Block List of IO CR Parameter Blocks List of Expected Submodule Blocks Alarm CR Parameter Block Parameter Server Block List of Multicast CR Block AR RPC Block IR Info Block SR Info Block AR Vendor Block AR Server Block AR FSU Block
CM_Dcontrol.ind (AREP, ControlBlock, ListOfSubmodules)	MAP_BOP_IND	Prm Begin.ind	AREP Session Key List of Submodules
CM_Dcontrol.ind (AREP, ControlBlock)	MAP_EOP_IND	Prm End.ind	AREP Session Key Alarm Sequence Number
CM_Read.ind (AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_IND	Read.ind	AREP API Target ARUUID Slot Number Subslot Number Index Seq Number Length

Primitive mapped from	Parameter mapped from	Primitive name	Associated parameters
CM_Write.ind (AREP, SlotNumber, SubslotNumber, Index, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_IND	Write.ind	AREP API Slot Number Subslot Number Index Prm Flag Seq Number Length Data
ALPMI_Alarm_Notification.cnf (+)(CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, Module_Ident_Number, Submodule_Ident_Number, Alarm_User_Data_Structure_Identifier, Alarm_User_Data, PNIO_Status)	MAP_AA_IND+	Alarm.cnf (+)	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier PNIO Status
ALPMI_Alarm_Notification.cnf (-)(CREP, ERRCLS, ERRCODE)	MAP_AN_CNF-	Alarm.cnf (-)	AREP Status
ALPMR_Alarm_Notification.ind (CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, Module_Ident_Number, Submodule_Ident_Number, User_Structure_Identifier, User_Data)	MAP_AN_IND	Alarm.ind	AREP API Alarm Priority Alarm Type Slot Number Subslot Number Alarm Specifier Module Ident Number Submodule Ident Number Alarm Item

5.5.3.3 State transition diagram

The FSPMDEV defines no state machine. Therefore no states are described.

5.5.3.4 State machine description

The FSPMDEV defines no state machine. Therefore no state machine description exists.

5.5.3.5 FSPMDEV state table

The FSPMDEV defines no state machine because all services are transferred to the underlying protocol machines.

5.5.3.6 Functions, Macros, Timers and Variables

Table 1052 shows the functions, macros, timers and variables defined for FSPMDEV, which are used by service primitives issued by the FAL user.

Table 1052 – Functions, Macros, Timers and Variables used by the AP-Context (FAL user) to FSPMDEV

Name	Function
MAP_ALARM_RSP-	if (PNIO Status != GOOD) then CREP=AREP.CREP API Alarm_Type=Alarm Type Slot_Number=Slot Number Subslot_Number=Subslot Number Alarm_Specifier=Alarm Specifier Sequence_Number PNIO_Status=PNIO Status
MAP_ALARM_RSP+	if (PNIO Status == GOOD) then CREP=AREP.CREP API Alarm_Type=Alarm Type Slot_Number=Slot Number Subslot_Number=Subslot Number Alarm_Specifier=Alarm Specifier Sequence_Number PNIO_Status=PNIO Status
MAP_AN_REQ	map service parameter 1:1 (=)
MAP_AREADY_REQ	AREP=AREP ControlBlockConnect.SessionKey=Session Key^(ControlBlockPlug.SessionKey=Session Key,ControlBlockPlug.AlarmSequenceNumber=Alarm Sequence Number) ModuleDiffBlock=Module Diff Block
MAP_BOP_RSP-	map service parameter 1:1 (=)
MAP_BOP_RSP+	AREP=AREP ControlBlockConnect.SessionKey=Session Key
MAP_CON_RSP-	map service parameter 1:1 (=)
MAP_CON_RSP+	AREP=AREP ARBlockRes=AR Response Block ListOfIOCRBlockRes=List of IO CR Response Blocks AlarmCRBlockRes=Alarm CR Response Block ModuleDiffBlock=Module Diff Block ARRPCBlock=AR RPC Block ARVendorBlock=AR Vendor Block
MAP_EOP_RSP-	map service parameter 1:1 (=)
MAP_EOP_RSP+	AREP=AREP ControlBlockConnect.SessionKey=Session Key^(ControlBlockPlug.SessionKey=Session Key,ControlBlockPlug.AlarmSequenceNumber=Alarm Sequence Number)
MAP_GINIOCS_REQ	CREP=CREP
MAP_OIOCS_REQ	CREP=CREP Data=IOCS
MAP_READ_RSP-	map service parameter 1:1 (=)
MAP_READ_RSP+	map service parameter 1:1 (=)
MAP_SIN_REQ	CREP=CREP Data={IOPS, Subslot Input Data}
MAP_SPS_REQ	CREP=AREP.CREP DataStatus={ProviderState=ProviderState Flag}
MAP_SRS_REQ	CREP=AREP.CREP DataStatus={Redundancy=Redundancy Flag}
MAP_SSS_REQ	CREP=AREP.CREP DataStatus={State=State Flag}
MAP_WRITE_RSP-	map service parameter 1:1 (=)
MAP_WRITE_RSP+	map service parameter 1:1 (=)

Table 1053 shows the functions, macros, timers and variables defined for FSPMDEV, which are used by service primitives issued by the FSPMDEV.

Table 1053 – Functions, Macros, Timers and Variables used by the FSPMDEV to AP-Context (FAL user)

Name	Function
MAP_AA_IND-	if (PNIO_Status != GOOD) then AREP=CREP.AREP API Alarm Type=Alarm_Type Slot Number=Slot_Number Subslot Number=Subslot_Number Alarm Specifier=Alarm_Specifier PNIO Status=PNIO_Status
MAP_AA_IND+	if (PNIO_Status == GOOD) then AREP=CREP.AREP API Alarm Type=Alarm_Type Slot Number=Slot_Number Subslot Number=Subslot_Number Alarm Specifier=Alarm_Specifier PNIO Status=PNIO_Status
MAP_AN_CNF-	AREP=CREP.AREP Status=(ERRCLS, ERRCODE)
MAP_AN_CNF+	AREP=CREP.AREP
MAP_AN_IND	AREP=CREP.AREP API Alarm Priority Alarm Type =Alarm_Type Slot Number =Slot_Number Subslot Number =Subslot_Number Alarm Specifier=Alarm_Specifier Module Ident Number=Module_Ident_Number Submodule Ident Number=Submodule_Ident_Number Alarm Item = {User_Structure_Identifier, User_Data}
MAP_AREADY_CNF-	map service parameter 1:1 (=)
MAP_AREADY_CNF+	AREP=AREP Session Key=ControlBlockPlug.SessionKey^ControlConnectPlug.SessionKey
MAP_BOP_IND	if (ControlBlockConnect.PrmEnd=TRUE) then AREP=AREP.CREP Session Key Alarm Sequence Number ControlCommand.PrmBegin=PrmBegin
MAP_CON_IND	AREP=AREP AR Parameter Block=ARBlockReq List of IO CR Parameter Blocks=ListOfIOCRBlockReq List of Expected Submodule Blocks=ListOfExpectedSubmoduleBlockReq Alarm CR Parameter Block=AlarmCRBlockReq Parameter Server Block=ParameterServerBlock List of Multicast CR Block=ListOfMulticastCRBlock AR RPC Block=ARRPCBlock IR Info Block=IRInfoBlock SR Info Block=SRInfoBlock AR Vendor Block=ARVendorBlock AR Server Block=ARServerBlock AR FSU Block=ARFSUBlock
MAP_DIAGEVT_IND	AREP =AREP CREP=CREP Alarm Item = Diagnosis Data
MAP_EOP_IND	if (List of Submodules=ListOfSubmodules) then AREP=AREP Session Key=ControlBlockConnect.SessionKey^ControlBlockPlug.SessionKey Alarm Sequence Number=ControlBlockPlug.AlarmSequenceNumber

Name	Function
MAP_GINIOCS_CNF-	AREP=AREP.CREP IOCS=Status RevCounter
MAP_GINIOCS_CNF+	AREP=AREP.CREP
MAP_GOUTD_CNF-	AREP=CREP.AREP other ignore
MAP_GOUTD_CNF+	AREP=CREP IOPS=Data Subslot Output Data=Data New Flag=New_Flag IOCS=Data
MAP_INDATA_IND	AREP=CREP.AREP CREP=CREP Slot Number =0 Subslot Number =0 InData Flag=TRUE
MAP_NEWDATA_IND	for each Slot/Subslot AREP=CREP.AREP CREP=CREP Slot Number=AREP.SlotNumber Subslot Number=AREP.SubslotNumber InData Flag=data local New Output.ind(AREP, CREP, Slot Number, Subslot Number, InData Flag)if (data != NoData) AREP=CREP.AREP CREP=CREP
MAP_NEWSTATUS_IND	if (APDU_Status has been changed) AREP=CREP.AREP CREP=CREP DataValid Flag=APDU_Status.DataStatus.DataValid State Flag=APDU_Status.DataStatus.State Redundancy Flag=APDU_Status.DataStatus.Redundancy ProviderState Flag=APDU_Status.DataStatus.ProviderState ProblemIndicator Flag=APDU_Status.DataStatus.ProblemIndicator
MAP_NODATA_IND	AREP=CREP.AREP CREP=CREP Slot Number =0 Subslot Number =0 Watchdog Flag=TRUE
MAP_OIOCS_CNF	AREP=CREP.AREP other ignore

MAP_READ_IND	if ((Index < =0x7FFF) OR (0xAFF0<=Index<=0xAFFF)) then
	AREP=AREP
	API=API
	Target ARUUID = TargetARUUID
	Slot Number = SlotNumber
	Subslot Number =SubslotNumber
	Index =Index
	Seq Number =SeqNumber
	Length=Length
	else
	map to special FAL user Read service primitive

MAP_REL_IND	AREP=AREP Session Key=ReleaseBlock.SessionKey
MAP_SIN_CNF	AREP=CREP.AREP other ignore
MAP_SPS_CNF	AREP=CREP.AREP other ignore
MAP_SYNCH_IND	Slot Subslot Global Cycle Counter=CycleCounter Phase Status

Name	Function
MAP_SYSTI_IND	Sync Data Sync Error Status=Status

MAP_WRITE_IND	if ((Index < =0x7FFF) OR (0xAFF0<=Index<=0xAFFF)) then
	AREP=AREP
	API=API
	Slot Number = SlotNumber
	Subslot Number =SubslotNumber
	Index =Index
	Multiple = Multiple
	Seq Number =SeqNumber
	Length=Length
	Data=Data
	else
	map to special FAL user Write service primitive

5.5.4 FAL Service Protocol Machine Controller

5.5.4.1 Overview

The FSPMCTL provides an interface for the user of the Controller component.

5.5.4.2 Primitive definitions

Table 1054 shows the service primitives including their associated parameters issued by the AP-Context (FAL user) and received by the FSPMCTL with the mapping to the underlying services.

Table 1054 – Primitives issued by AP-Context (FAL user) to FSPMCTL

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
local AR Abort.req	AREP Session Key	CM_Release.req(AREP, ControlBlock)	MAP_REL_REQ
local Create LogBook Entry	info	CreateLogBookEntry (info)	info = info
local Get Input.req	AREP CREP Slot Number Subslot Number	CPM_Get_Data_req (CREP)	MAP_GIN_REQ
local Get Output IOCS.req	AREP CREP Slot Number Subslot Number	CPM_Get_Status_req (CREP)	MAP_GOUTIOCS_REQ
local Set AR State.req	AREP ARstate	Set_Variable (ARstate)	AREP=AREP ARstate=ARstate
local Set Input IOCS.req	AREP CREP Slot Number Subslot Number IOCS	PPM_Set_Data_req (CREP, Data)	MAP_SINIOCS_REQ
local Set Output.req	AREP CREP Slot Number Subslot Number IOPS Subslot Output Data	PPM_Set_Data_req (CREP, Data)	MAP_SOUT_REQ
local Set Provider State.req	AREP Provider State Flag	PPM_Set_Status_req (CREP, D_Status)	MAP_SPS_REQ

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
local Set Redundancy.req	AREP Redundancy Flag	PPM_Set_Status_req (CREP, D_Status)	MAP_SRS_REQ
local Set State.req	AREP State Flag	PPM_Set_Status_req (CREP, D_Status)	MAP_SSS_REQ
Application Ready.rsp(-)	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Ccontrol.rsp (-)(AREP, ControlBlock)	MAP_AREADY_RSP-
Application Ready.rsp(+)	AREP Session Key Alarm Sequence Number	CM_Ccontrol.rsp (+)(AREP, ControlBlock)	MAP_AREADY_RSP+
Connect.req	AREP AR Parameter Block List of IO CR Parameter Blocks List of Expected Submodule Blocks Alarm CR Parameter Block Parameter Server Block List of Multicast CR Blocks AR RPC Block IR Info Block SR Info Block AR Vendor Block AR Server Block AR FSU Block	CM_Connect.req (AREP ARBlockReq ListOfIOCRBlockReq ListOfExpectedSubmoduleBlockReq AlarmCRBlockReq ParameterServerBlock ListOfMulticastCRBlock ARRPCBlock IRInfosBlock SRInfoBlock ARVendorBlock ARServerBlock ARFSUBlock)	MAP_CON_REQ
Prm Begin.req	AREP Session Key Alarm Sequence Number List Of Submodules	CM_Dcontrol.req (AREP, ControlBlock, ListOfSubmodules)	MAP_PB_REQ
Prm End.req	AREP Session Key Alarm Sequence Number	CM_Dcontrol.req (AREP, ControlBlock)	MAP_EOP_REQ
Read.req	AREP API Target ARUUID Slot Number Subslot Number Index Seq Number Length	CM_Read.req (AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_REQ
Read.rsp (-)	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_RSP-

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
Read.rsp (+)	AREP Seq Number Length Data	CM_Read.rsp (+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RSP+
Write.req	AREP API Slot Number Subslot Number Index Seq Number Length Data	CM_Write.req (AREP, SlotNumber, SubslotNumber, Index, SeqNumber, Length, Data)	MAP_WRITE_REQ
Write.rsp (-)	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp (-)(AREP, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_RSP-
Write.rsp (+)	AREP Seq Number	CM_Write.rsp (+) (AREP, Seq Number, AddData1, AddData2)	MAP_WRITE_RSP+
Alarm.req	AREP API Alarm Priority Alarm Type Slot Number Subslot Number Alarm Specifier Module Ident Number Submodule Ident Number Alarm Item	ALPMI_Alarm_Notification.req (CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, Module_Ident_Number, Submodule_Ident_Number, Alarm_User_Data_Structure_Identifier, Alarm_User_Data)	MAP_AN_REQ
Alarm.rsp (-)	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier PNIO Status	ALPMR_Alarm_Ack.req (CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, PNIO_Status)	MAP_ALARM_RSP-
Alarm.rsp (+)	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier PNIO Status	ALPMR_Alarm_Ack.req (CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, PNIO_Status)	MAP_ALARM_RSP+

Table 1055 shows the service primitives including their associated parameters issued by the FSPMCTL and received by the AP-Context (FAL user).

Table 1055 – Primitives issued by FSPMCTL to AP-Context (FAL user)

Primitive mapped to	Parameter mapped from	Primitive name	Associated parameters
CM_Abort.cnf (-)	AREP=AREP	local AR Abort.cnf (-)	AREP
CM_Abort.cnf (+)	AREP=AREP	local AR Abort.cnf (+)	AREP
CM_Abort.ind	AREP	local AR Abort.ind	AREP
CMCTL_state_ind (AREP, state)	if (state == DATA) then AREP=AREP else ignore	local AR In Data.ind	AREP
CPM_NewData_ind (AREP, CREP, APDU_Status)	MAP_NEWSTATUS_IND	local Data State Changed.ind	AREP CREP DataValid Flag State Flag Redundancy Primary State Flag ProviderState Flag ProblemIndicator Flag
DiagnosisEvent (AREP, CREP, API, Diagnosis Data)	MAP_DIAGEVT_IND	local Diagnosis Event.ind	AREP CREP Alarm Item
CPM_Get_Data_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_GIND_CNF-	local Get Input.cnf(-)	AREP
CPM_Get_Data_cnf (+) (CREP, Data, New_Flag)	MAP_GIND_CNF+	local Get Input.cnf(+)	AREP IOPS Subslot Input Data New Flag IOCS
CPM_Get_Status_cnf (-) (CREP, ERRCLS, ERRCODE)	MAP_GOUTIOCS_CNF-	local Get Output IOCS.cnf(-)	AREP
CPM_Get_Status_cnf (+) (CREP, Status, RecvCounter)	MAP_GOUTIOCS_CNF+	local Get Output IOCS.cnf(+)	AREP IOCS
CPM_NewData_ind (AREP, CREP, APDU_Status)	MAP_NEWDATA_IND	local New Input.ind	AREP CREP Slot Number Subslot Number InData Flag
if (Set_Variable (ARstate)) == FALSE	AREP=AREP	local Set AR State.cnf (-)	AREP
if (Set_Variable (ARstate)) == TRUE	AREP=AREP	local Set AR State.cnf (+)	AREP
PPM_Set_Data_cnf(-) (CREP, ERRCLS, ERRCODE)	MAP_IIOCS_CNF	local Set Input IOCS.cnf(-)	AREP
PPM_Set_Data_cnf(+) (CREP)	MAP_IIOCS_CNF	local Set Input IOCS.cnf(+)	AREP
PPM_Set_Data_cnf(-) (CREP, ERRCLS, ERRCODE)	MAP_IIOCS_CNF	local Set Output.cnf(-)	AREP

Primitive mapped to	Parameter mapped from	Primitive name	Associated parameters
PPM_Set_Data_cnf(+) (CREP)	MAP_IIOCS_CNF	local Set Output.cnf(+)	AREP
PPM_Set_Status_cnf(-) (CREP, ERRCLS, ERRCODE)	MAP_SPS_CNF	local Set Provider State.cnf(-)	AREP
PPM_Set_Status_cnf(+) (CREP)	MAP_SPS_CNF	local Set Provider State.cnf(+)	AREP
PPM_Set_Status_cnf(-) (CREP, ERRCLS, ERRCODE)	MAP_SRS_CNF	local Set Redundancy.cnf(-)	AREP
PPM_Set_Status_cnf(+) (CREP)	MAP_SRS_CNF	local Set Redundancy.cnf(+)	AREP
PPM_Set_Status_cnf(-) (CREP, ERRCLS, ERRCODE)	MAP_SPS_CNF	local Set State.cnf(-)	AREP
PPM_Set_Status_cnf(+) (CREP)	MAP_SPS_CNF	local Set State.cnf(+)	AREP
SyncStateChange_ind (Status)	MAP_SYSTI_IND	local Sync State Info.ind	Sync Data Sync Error Status
TickEvent (CycleCounter, CycleOffset)	MAP_SYNCH_IND	local SYNCH Event.ind	Slot Subslot Global Cycle Counter Phase Status
CM_Ccontrol.ind (AREP, ControlBlock, ModuleDiffBlock)	MAP_AREADY_IND	Application Ready.ind	AREP Session Key Alarm Sequence Number Module Diff Block
CM_Connect.cnf (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_CON_CNF-	Connect.cnf(-)	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2
CM_Connect.cnf (+)(AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock, ARRPCBlock, ARVendorBlock)	MAP_CON_CNF+	Connect.cnf(+)	AREP AR Response Block List of IO CR Response Blocks Alarm CR Response Block Module Diff Block AR RPC Block AR Vendor Block
CM_Release.cnf (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_REL_CNF-	local AR Abort.cnf (-)	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2
CM_Release.cnf (+)(AREP, ControlBlock)	MAP_REL_CNF+	local AR Abort.cnf (+)	AREP Session Key

Primitive mapped to	Parameter mapped from	Primitive name	Associated parameters
CM_Dcontrol.cnf (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	AREP	Prm Begin.cnf(-)	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2
CM_Dcontrol.cnf (+) (AREP, ControlBlock)	AREP	Prm Begin.cnf(+)	AREP Session Key
CM_Dcontrol.cnf (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_EOP_CNF-	Prm End.cnf(-)	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2
CM_Dcontrol.cnf (+) (AREP, ControlBlock)	MAP_EOP_CNF+	Prm End.cnf(+)	AREP Session Key Alarm Sequence Number
CM_Read.cnf (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_CNF-	Read.cnf(-)	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2
CM_Read.cnf (+)(AREP, Seq Number, Length, Data)	MAP_READ_CNF+	Read.cnf(+)	AREP Seq Number Length Data
CM_Read.ind (AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_IND	Read.ind	AREP API Target ARUUID Slot Number Subslot Number Index Seq Number Length
CM_Write.cnf (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_CNF-	Write.cnf(-)	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2
CM_Write.cnf (+)(AREP, Seq Number)	MAP_WRITE_CNF+	Write.cnf(+)	AREP Seq Number
CM_Write.ind (AREP, SlotNumber, SubslotNumber, Index, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_IND	Write.ind	AREP API Slot Number Subslot Number Index Prm Flag Seq Number Length Data

Primitive mapped to	Parameter mapped from	Primitive name	Associated parameters
ALPMI_Alarm_Notification.cnf (+)(CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, Module_Ident_Number, Submodule_Ident_Number, Alarm_User_Data_Structure_Identifier, Alarm_User_Data, PNIO_Status)	MAP_AA_IND+	Alarm.cnf (+)	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier PNIO Status
ALPMI_Alarm_Notification.cnf (-)(CREP, ERRCLS, ERRCODE)	MAP_AN_CNF	Alarm.cnf(-)	AREP Status
ALPMR_Alarm_Notification.ind (CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, Module_Ident_Number, Submodule_Ident_Number, User_Structure_Identifier, User_Data)	MAP_AN_IND	Alarm.ind	AREP API Alarm Priority Alarm Type Slot Number Subslot Number Alarm Specifier Module Ident Number Submodule Ident Number Alarm Item

5.5.4.3 State transition diagram

The FSPMCTL defines no state machine. Therefore no states are described.

5.5.4.4 State machine description

The FSPMCTL defines no state machine. Therefore no state machine description exists.

5.5.4.5 FSPMCTL state table

The FSPMCTL defines no state machine because all services are transferred to the underlying protocol machines.

5.5.4.6 Functions, Macros, Timers and Variables

Table 1056 shows the functions, macros, timers and variables defined for FSPMCTL, which are used by service primitives issued by the FAL user.

Table 1056 – Functions, Macros, Timers and Variables used by AP-Context (FAL user) to FSPMCTL

Name	Function
MAP_ALARM_RSP-	if (PNIO Status != GOOD) then CREP=AREP.CREP API Alarm_Type=Alarm Type Slot_Number=Slot Number Subslot_Number=Subslot Number Alarm_Specifier=Alarm Specifier Sequence_Number PNIO_Status=PNIO Status
MAP_ALARM_RSP+	if (PNIO Status == GOOD) CREP=AREP.CREP API Alarm_Type=Alarm Type Slot_Number=Slot Number Subslot_Number=Subslot Number Alarm_Specifier=Alarm Specifier Sequence_Number PNIO_Status=PNIO Status
MAP_AN_REQ	CREP=AREP.CREP API Alarm Priority Alarm_Type=Alarm Type Slot_Number=Slot Number Subslot_Number=Subslot Number Alarm_Specifier=Alarm Specifier Module_Ident_Number=Module Ident Number Submodule_Ident_Number=Submodule Ident Number Alarm_User_Data_Structure_Identifier=User Structure Identifier Alarm_User_Data=User Structure Identifier.User Data
MAP_AREADY_RSP-	AREP=AREP Error Decode Error Code 1 Error Code 2
MAP_AREADY_RSP+	AREP=AREP ControlBlockConnect.SessionKey=Session Key^(ControlBlockPlug.SessionKey=Session Key,ControlBlockPlug.AlarmSequenceNumber=Alarm Sequence Number)
MAP_CON_REQ	AREP=AREP ARBlockReq=AR Parameter Block ListOfIOCRBlockReq=List of IO CR Parameter Blocks ListOfExpectedSubmoduleBlockReq=List of Expected Submodule Blocks AlarmCRBlockReq=Alarm CR Parameter Block ParameterServerBlock=Parameter Server Block ListOfMulticastCRBlock=List of Multicast CR Block ARRPCBlock=AR RPC Block IRInfoBlock=IR Info Block SRInfoBlock=SR Info Block ARVendorBlock=AR Vendor Block ARServerBlock=AR Server Block AR FSU Block=AR FSU Block
MAP_EOP_REQ	AREP=AREP ControlBlockConnect.SessionKey=Session Key^(ControlBlockPlug.SessionKey=Session Key,ControlBlockPlug.AlarmSequenceNumber=Alarm Sequence Number) ControlBlockConnect.PrmEnd=TRUE
MAP_GIN_REQ	CREP=CREP
MAP_GOUTIOCS_REQ	CREP=CREP
MAP_PB_REQ	AREP=AREP ControlBlockConnect.SessionKey=Session Key ControlCommand.PrmBegin=PrmBegin ListOfSubmodules=List Of Submodules

Name	Function
MAP_READ_REQ	map service parameter 1:1 (=)
MAP_READ_RSP-	map service parameter 1:1 (=)
MAP_READ_RSP+	map service parameter 1:1 (=)
MAP_REL_REQ	AREP=AREP ReleaseBlock.SessionKey=SessionKey
MAP_SINIOCS_REQ	CREP=CREP Data=IOCS
MAP_SOUT_REQ	CREP=CREP Data={IOPS,Subslot Output Data}
MAP_SPS_REQ	CREP=AREP.CREP DataStatus={ProviderState=ProviderState Flag}
MAP_SRS_REQ	CREP=AREP.CREP DataStatus={Redundancy=Redundancy Flag}
MAP_SSS_REQ	CREP=AREP.CREP DataStatus={State=State Flag}
MAP_WRITE_REQ	AREP=AREP API SlotNumber=Slot Number SubslotNumber=Subslot Number Index=Index Multiple=Multiple SeqNumber=Seq Number Length=Length Data=Data
MAP_WRITE_RSP-	map service parameter 1:1 (=)
MAP_WRITE_RSP+	map service parameter 1:1 (=)

Table 1057 shows the functions, macros, timers and variables defined for FSPMCTL, which are used by service primitives issued by the FSPMCTL.

Table 1057 – Functions, Macros, Timers and Variables used by FSPMCTL to AP-Context (FAL user)

Name	Function
MAP_AA_CNF	AREP=CREP.AREP other ignore
MAP_AA_IND-	if (PNIO_Status != GOOD) then AREP=CREP.AREP API Alarm Type=Alarm_Type Slot Number=Slot_Number Subslot Number=Subslot_Number Alarm Specifier=Alarm_Specifier PNIO Status=PNIO_Status
MAP_AA_IND+	if (PNIO_Status == GOOD) then AREP=CREP.AREP API Alarm Type=Alarm_Type Slot Number=Slot_Number Subslot Number=Subslot_Number Alarm Specifier=Alarm_Specifier PNIO Status=PNIO_Status
MAP_AN_CNF	AREP=CREP.AREP Status

Name	Function
MAP_AN_IND	AREP=CREP.AREP API Alarm Priority Alarm Type=Alarm_Type Slot Number=Slot_Number Subslot Number=Subslot_Number Alarm Specifier=Alarm_Specifier Module Ident Number=Module_Ident_Number Submodule Ident Number=Submodule_Ident_Number Alarm Item={User_Structure_Identifier, User_Data}
MAP_AREADY_IND	AREP=AREP Session Key=ControlBlockPlug.SessionKey^ControlConnectPlug.SessionKey Module Diff Block=ModuleDiffBlock
MAP_CON_CNF-	AREP=AREP Error Decode=ErrorDecode Error Code 1=ErrorCode1 Error Code 2=ErrorCode2 Add Data 1=AddData1 Add Data 2=AddData2
MAP_CON_CNF+	AREP=AREP AR Response Block=ARBlockRes List of IO CR Response Blocks=ListOfIOCRBlockRes Alarm CR Response Block=AlarmCRBlockRes Module Diff Block=ModuleDiffBlock AR RPC Block=ARRPCBlock AR Vendor Block=ARVendorBlock
MAP_DIAGEVT_IND	AREP=AREP CREP=CREP Alarm Item=Diagnosis Data
MAP_EOP_CNF-	AREP=AREP Error Decode=ErrorDecode Error Code 1=ErrorCode1 Error Code 2=ErrorCode2 Add Data 1=AddData1 Add Data 2=AddData2
MAP_EOP_CNF+	AREP=AREP Session Key=ControlBlockConnect.SessionKey^ControlBlockPlug.SessionKey Alarm Sequence Number=ControlBlockPlug.AlarmSequenceNumber
MAP_GIND_CNF-	AREP=CREP.AREP other ignore
MAP_GIND_CNF+	AREP=CREP IOPS=Data Subslot Input Data=Data New Flag=New_Flag IOCS=Data
MAP_GOUTIOCS_CNF-	AREP=CREP.AREP ERRCLS ERRCODE
MAP_GOUTIOCS_CNF+	AREP=CREP.AREP IOCS=Status RevCounter
MAP_IIOCS_CNF	AREP=CREP.AREP other ignore
MAP_INDATA_IND	AREP=CREP.AREP CREP=CREP Slot Number=0 Subslot Number=0 InData Flag=TRUE
MAP_NEWDATA_IND	if (Data has been changed) then AREP=AREP CREP=CREP Slot Number Subslot Number

Name	Function
MAP_NEWSTATUS_IND	if (APDU_Status has been changed) AREP=CREP.AREP CREP=CREP DataValid Flag=APDU_Status.DataStatus.DataValid State Flag=APDU_Status.DataStatus.State Redundancy Flag=APDU_Status.DataStatus.Redundancy ProviderState Flag=APDU_Status.DataStatus.ProviderState ProblemIndicator Flag=APDU_Status.DataStatus.ProblemIndicator
MAP_NODATA_IND	AREP=CREP.AREP CREP=CREP Slot Number=0 Subslot Number=0 Watchdog Flag=TRUE
MAP_READ_CNF-	if (ServiceReqWasRead(AREP, Seq Number) == TRUE) then AREP=AREP Seq Number=Seq Number Error Decode=ErrorDecode Error Code 1=ErrorCode1 Error Code 2=ErrorCode2 Add Data 1=AddData1 Add Data 2=AddData2
MAP_READ_CNF+	if (ServiceReqWasRead(AREP, Seq Number) == TRUE) then AREP=AREP Seq Number=Seq Number Length =Length Data=Data

MAP_READ_IND	if ((Index <=0x7FFF) OR (0xAFF0<=Index<=0xAFFF)) then
	AREP=AREP
	API=API
	Target ARUUID=TargetARUUID
	Slot Number=SlotNumber
	Subslot Number=SubslotNumber
	Index=Index
	Seq Number=SeqNumber
	Length=Length
	else
	map to special FAL user Read service primitive

MAP_REL_CNF-	AREP=AREP Error Decode=ErrorDecode Error Code 1=ErrorCode1 Error Code 2=ErrorCode2
MAP_REL_CNF+	AREP=AREP Session Key=ReleaseBlock.SessionKey
MAP_SPS_CNF	AREP=CREP.AREP other ignore
MAP_SRS_CNF	AREP=CREP.AREP other ignore
MAP_SYNCH_IND	Slot Subslot Global Cycle Counter=CycleCounter Phase Status
MAP_SYSTI_IND	Sync Data Sync Error Status=Status
MAP_WRITE_CNF-	if (ServiceReqWasWrite(AREP, Seq Number) == TRUE) then AREP=AREP Multiple Seq Number Error Decode=ErrorDecode Error Code 1=ErrorCode1 Error Code 2=ErrorCode2 Add Data 1=AddData1 Add Data 2=AddData2

Name	Function
MAP_WRITE_CNF+	if (ServiceReqWasWrite(AREP, Seq Number) == TRUE) then AREP=AREP Multiple Seq Number=SeqNumber

MAP_WRITE_IND	if ((Index <= 0x7FFF) OR (0xAFF0 <= Index <= 0xAFFF)) then
	AREP=AREP
	API=API
	Slot Number=SlotNumber
	Subslot Number=SubslotNumber
	Index=Index
	Multiple=Multiple
	Seq Number=SeqNumber
	Length=Length
	Data=Data
	else
	map to special FAL user Write service primitive

5.5.5 FAL Service Protocol Machine Network Management Entity

5.5.5.1 Overview

The FSPMNME provides an interface for the user of the Network Management Entity component.

5.5.5.2 Primitive definitions

Table 1058 shows the service primitives including their associated parameters issued by the AP-Context (FAL user) and received by the FSPMNME with the mapping to the underlying services.

Table 1058 – Primitives issued by AP-Context (FAL user) to FSPMNME

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
–	–	–	–

Table 1059 shows the service primitives including their associated parameters issued by the FSPMNME and received by the AP-Context (FAL user).

Table 1059 – Primitives issued by FSPMNME to AP-Context (FAL user)

Primitive mapped from	Parameter mapped from	Primitive name	Associated parameters
–	–	–	–

5.5.5.3 State transition diagram

The FSPMNME defines no state machine. Therefore, no states are described.

5.5.5.4 State machine description

The FSPMNME defines no state machine. Therefore, no state machine description exists.

5.5.5.5 FSPMNME state table

The FSPMNME defines no state machine because all services are transferred to the underlying protocol machines.

5.5.5.6 Functions, Macros, Timers and Variables

The FSPMNME defines no functions, macros, timers, and variables. Therefore, no description exists.

5.6 Application Relationship Protocol Machines

5.6.1 Alarm Protocol Machine Initiator

5.6.1.1 Primitive definitions

5.6.1.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Alarm Protocol Machine Initiator (ALPMI) are described in the service definition and shown in Table 1060.

Table 1060 – Remote primitives issued or received by ALPMI

Primitive	Source	Destination	Associated parameters	Functions
APMR_A_Data.ind	APMR	ALPMI	CREP, Data	—
APMS_A_Data.cnf (-)	APMS	ALPMI	CREP, ERRCLS, ERRCODE	—
APMS_A_Data.cnf (+)	APMS	ALPMI	CREP	—
ALPMI_Alarm_Notification.req	FSPMDEV FSPMCTL	ALPMI	CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, Module_Ident_Number, Submodule_Ident_Number, Alarm_User_Data_Structure_Identifier, Alarm_User_Data	The service Alarm_Notification conveys alarm data.
APMS_A_Data.req	ALPMI	APMS	CREP, Data	—
ALPMI_Alarm_Notification.cnf (-)	ALPMI	FSPMDEV FSPMCTL	CREP, ERRCLS, ERRCODE	This service primitive indicates that the Alarm_Notification service failed.
ALPMI_Alarm_Notification.cnf (+)	ALPMI	FSPMDEV FSPMCTL	CREP	This service primitive indicates that the Alarm_Notification service succeeded. PNIOStatus: – “No Error” – “Alarm Type Not Supported” – “Wrong Submodule State” – “IOCARSR: Backup – Alarm not executed”

5.6.1.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by ALPMI are described in the service definition and shown in Table 1061.

Table 1061 – Local primitives issued or received by ALPMI

Primitive	Source	Destination	Associated parameters	Functions
ALPMI_Activate_req	CMSU CTLSU	ALPMI	CREP, DA, SA, VLAN, RTATimeoutFactor, RTARetries	The service Activate initializes the ALPMI and requests the initialization of the APMS and APMR protocol machines.
ALPMI_Close_req	CMSU CTLSU	ALPMI	CREP	The service Close deinitializes the ALPMI and closes the APMR and APMS protocol machines.
ALPMI_Activate_cnf (-)	ALPMI	CMSU CTLSU	CREP, ERRCLS, ERRCODE	—
ALPMI_Activate_cnf (+)	ALPMI	CMSU CTLSU	CREP	—
ALPMI_Close_cnf (+)	ALPMI	CMSU CTLSU	CREP	—
ALPMI_Error_ind	ALPMI	CMSU CTLSU	CREP, ERRCLS, ERRCODE	This service primitive indicates a failure during transmission of alarm data.

5.6.1.2 State transition diagram

The state transition diagram of the ALPMI is shown in Figure 195.

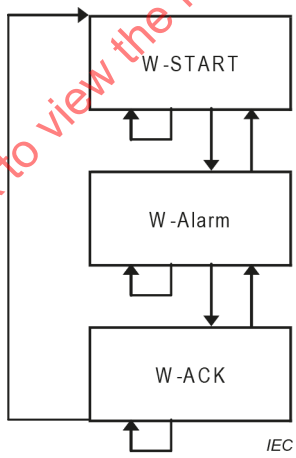


Figure 195 – State transition diagram of ALPMI

States of the ALPMI are:

W-START The W-START state indicates that the initialization is needed. The Activate service sets the machine to the W-Alarm state.

W-Alarm Indicates a successful initialization and the state machine is waiting for Alarm Notification requests

W-ACK Wait for an Alarm Acknowledge and generate the Alarm Notification confirmation.

5.6.1.3 State machine description

This state machine handles the Alarm Notification. Each priority (High and Low) is handled independently. An application issues an Alarm.req which is conveyed by the ALPMI and the APMS to the APMR and the ALPMR. The ALPMR indicates the Alarm.ind to the remote application.

The remote application issues an Alarm.rsp to the ALPMR and the APMS which is conveyed to the APMR and the ALPMI. The ALPMI issues an Alarm.cnf to the application.

5.6.1.4 ALPMI state table

Table 1062 contains the description of the ALPMI state machine.

Table 1062 – ALPMI state table

#	Current State	Event /Condition =>Action	Next State
1	W-START	**ALPMI_Activate_req ()** => AlarmInstance[SrcSAP, DstSAP, Prio] := DA, RTATimeoutFactor, RTARetries, … //NOTE Information used for ALPMI, ALPMR, APMS and APMR Use CREP to identify the AlarmInstance ALPMI_Activate_cnf (+)	W-Alarm
2	W-START	**ALPMI_Close_req ()** => ALPMI_Close_cnf ()	W-START
3	W-START	**APMS_A_Data.cnf (+)** => ignore	W-START
4	W-START	**APMS_A_Data.cnf (-)** => ERRCLS := ALPMI ERRCODE := Invalid ALPMI_Error_ind ()	W-START
5	W-START	**ALPMI_Alarm_Notification.req ()** => ALPMI_Alarm_Notification.cnf (-)	W-START
6	W-START	**APMR_A_Data.ind ()** => ignore	W-START
7	W-Alarm	**ALPMI_Alarm_Notification.req ()** => Select the addressed / associated APMS and combine data for the alarm notification request APMS_A_Data.req ()	W-ACK
8	W-Alarm	**ALPMI_Activate_req ()** => ERRCLS:= ALPMI ERRCODE:= WRONG-STATE ALPMI_Activate_cnf (-)	W-Alarm
9	W-Alarm	**ALPMI_Close_req ()** => ALPMI_Close_cnf ()	W-START
10	W-Alarm	**APMS_A_Data.cnf (+)** => ignore	W-Alarm

#	Current State	Event /Condition =>Action	Next State
11	W-Alarm	**APMS_A_Data.cnf (-)** => ERRCLS := ALPMI ERRCODE := Invalid ALPMI_Error_ind ()	W-Alarm
12	W-Alarm	**APMR_A_Data.ind ()** => ignore	W-Alarm
13	W-ACK	**APMR_A_Data.ind ()** /Data == AlarmAck-PDU => ALPMI_Alarm_Notification.cnf (+)	W-Alarm
14	W-ACK	**ALPMI_Activate_req ()** => ERRCLS:= ALPMI ERRCODE:= WRONG-STATE ALPMI_Activate_cnf (-)	W-ACK
15	W-ACK	**ALPMI_Alarm_Notification.req ()** => ALPMI_Alarm_Notification.cnf (-)	W-ACK
16	W-ACK	**ALPMI_Close_req ()** => ALPMI_Close_cnf ()	W-START
17	W-ACK	**APMS_A_Data.cnf (+)** => ignore	W-ACK
18	W-ACK	**APMS_A_Data.cnf (-)** => ERRCLS := ALPMI ERRCODE := Invalid ALPMI_Error_ind ()	W-ACK
19	W-ACK	**APMR_A_Data.ind ()** /Data != AlarmAck-PDU => ERRCLS:= ALPMI ERRCODE:= WRONG-ACK-PDU ALPMI_Error_ind ()	W-Alarm

5.6.1.5 Functions, Macros, Timers and Variables

Table 1063 contains the functions, macros, timers and variables used by the ALPMI and their arguments and their descriptions.

Table 1063 – Functions, Macros, Timers and Variables used by ALPMI

Name	Type	Meaning
AlarmInstance	Variable	This AR global variable contains the required information for this protocol machine instance.
ERRCLS	Variable	This local variable contains the entity signaling the error.
ERRCODE	Variable	This local variable contains the error reason in the context of the ERRCLS.

5.6.2 Alarm Protocol Machine Responder

5.6.2.1 Primitive definitions

5.6.2.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Alarm Protocol Machine Responder (ALPMR) are described in the service definition and shown in Table 1064.

Table 1064 – Remote primitives issued or received by ALPMR

Primitive	Source	Destination	Associated parameters	Functions
APMR_A_Data.ind	APMR	ALPMR	CREP.APMR, Data	—
APMS_A_Data.cnf (-)	APMS	ALPMR	CREP.APMS, ERRCLS, ERRCODE	—
APMS_A_Data.cnf (+)	APMS	ALPMR	CREP	—
ALPMR_Alarm_Ack.req	FSPMCTL FSPMDEV	ALPMR	CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, PNIO_Status	The service Alarm_Ack conveys the alarm acknowledgement.
ALPMR_Alarm_Ack.cnf (-)	ALPMR	FSPMCTL FSPMDEV	CREP, ERRCLS, ERRCODE	This service primitive indicates that the Alarm_Ack service failed.
ALPMR_Alarm_Ack.cnf (+)	ALPMR	FSPMCTL FSPMDEV	CREP	This service primitive indicates that the Alarm_Ack service succeeded.
ALPMR_Alarm_Notification.ind	ALPMR	FSPMCTL FSPMDEV	CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, Module_Ident_Number, Submodule_Ident_Number, User_Structure_Identifier, User_Data	The service Alarm_Notification indicates alarm data.

5.6.2.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by ALPMR are described in the service definition and shown in Table 1065.

Table 1065 – Local primitives issued or received by ALPMR

Primitive	Source	Destination	Associated parameters	Functions
ALPMR_Activate_req	CMSU CTLSU	ALPMR	CREP, DA, SA, VLAN, RTATimeoutFactor, RTARetries	The service Activate initializes the ALPMR and requests the initialization of the APMS and APMR protocol machines.
ALPMR_Close_req	CMSU CTLSU	ALPMR	CREP	The service Close deinitializes the ALPMI and closes the APMR and APMS protocol machines.
ALPMR_Activate_cnf (-)	ALPMR	CMSU CTLSU	CREP, ERRCLS, ERRCODE	This service primitive indicates that the Activate service failed.
ALPMR_Activate_cnf (+)	ALPMR	CMSU CTLSU	CREP	This service primitive indicates that the Activate service succeeded.
ALPMR_Close_cnf (+)	ALPMR	CMSU CTLSU	CREP	This service primitive indicates that the Close service succeeded.
ALPMR_Error_ind	ALPMR	CMSU CTLSU	CREP, ERRCLS, ERRCODE	This service primitive indicates a failure during transmission of alarm data.

5.6.2.2 State transition diagram

The state transition diagram of the ALPMR is shown in Figure 196.

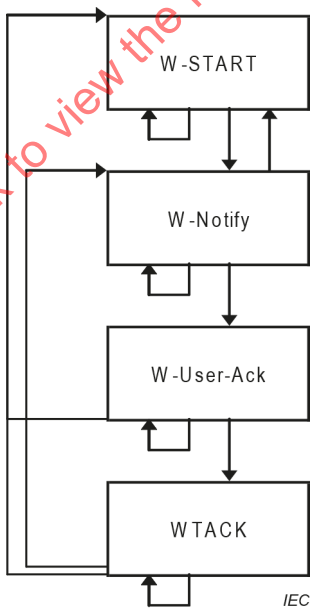


Figure 196 – State transition diagram of ALPMR

States of the ALPMR are:

W-START	The W-START state indicates that the initialization is needed. The Activate service sets the machine to the W-Notify state.
W-Notify	After successful initialization the state machine is waiting for Alarm Notification PDUs in the state W-Notify and afterwards enters the W-User-Ack state waiting for an Alarm Ack from the application.
W-User-Ack	Waiting for an Alarm Ack from the application.
WTACK	Wait for the confirmation of the transport acknowledge.

5.6.2.3 State machine description

This state machine handles the responder for an Alarm Notification. Each priority (High and Low) is handled independently.

A remote application issues an Alarm.req which is conveyed by the ALPMI and the APMS to the APMR and the ALPMR. The ALPMR indicates the Alarm.ind to the application.

The application issues an Alarm.rsp to the ALPMR and the APMS which is conveyed to the APMR and the ALPMI. The ALPMI issues an Alarm.cnf to the remote application.

5.6.2.4 ALPMR state table

Table 1066 contains the description of the ALPMR state machine.

Table 1066 – ALPMR state table

#	Current State	Event /Condition =>Action	Next State
1	W-START	**ALPMR_Activate_req ()** => AlarmInstance[SrcSAP, DstSAP, Prio] := DA, RTATimeoutFactor, RTARetries, … //NOTE Information used for ALPMI, ALPMR, APMS and APMR Use CREP to identify the AlarmInstance ALPMR_Activate_cnf (+)	W-Notify
2	W-START	**ALPMR_Close_req ()** => ALPMR_Close_cnf ()	W-START
3	W-START	**ALPMR_Alarm_Ack.req ()** => ERRCLS:= ALPMR ERRCODE:= WRONG-STATE ALPMR_Alarm_Ack.cnf (-)	W-START
4	W-START	**APMR_A_Data.ind ()** => ignore	W-START
5	W-START	**APMS_A_Data.cnf ()** => ignore	W-START
6	W-Notify	**ALPMR_Activate_req ()** => ERRCLS:= ALPMR ERRCODE:= WRONG-STATE ALPMR_Activate_cnf (-)	W-Notify

#	Current State	Event /Condition =>Action	Next State
7	W-Notify	**ALPMR_Alarm_Ack.req ()** => ERRCLS:= ALPMR ERRCODE:= WRONG-STATE ALPMR_Alarm_Ack.cnf (-)	W-Notify
8	W-Notify	**ALPMR_Close_req ()** => ALPMR_Close_cnf () ALPMR_Error_ind ()	W-START
9	W-Notify	**APMR_A_Data.ind ()** /Data != Alarm-Notification-PDU => ERRCLS:= ALPMR ERRCODE:= WRONG-NOTIFICATION-PDU ALPMR_Error_ind ()	W-Notify
10	W-Notify	**APMR_A_Data.ind ()** /Data = Alarm-Notification-PDU => Combine data for the alarm notification indication ALPMR_Alarm_Notification.ind ()	W-User-Ack
11	W-Notify	**APMS_A_Data.cnf (+)** => ignore	W-Notify
12	W-Notify	**APMS_A_Data.cnf (-)** => ERRCLS:= ALPMR ERRCODE:= Invalid ALPMR_Error_ind ()	W-Notify
13	W-User-Ack	**ALPMR_Activate_req ()** => ERRCLS:= ALPMR ERRCODE:= WRONG-STATE ALPMR_Activate_cnf (-)	W-User-Ack
14	W-User-Ack	**ALPMR_Alarm_Ack.req ()** => Select the addressed / associated APMS and combine data for the alarm ack request APMS_A_Data.req ()	WTACK
15	W-User-Ack	**ALPMR_Close_req ()** => ALPMR_Close_cnf () ALPMR_Error_ind ()	W-START
16	W-User-Ack	**APMR_A_Data.ind ()** => ERRCLS:= RTA_ERR_CLS_PROTOCOL ERRCODE:= AR protocol violation Combine data for the RTA error request ALPMR_Error_ind ()	W-User-Ack
17	W-User-Ack	**APMS_A_Data.cnf (+)** => ignore	W-User-Ack
18	W-User-Ack	**APMS_A_Data.cnf (-)** => ERRCLS:= ALPMR ERRCODE:= Invalid ALPMR_Error_ind ()	W-User-Ack
19	WTACK	**APMS_A_Data.cnf (+)** => ALPMR_Alarm_Ack.cnf (+)	W-Notify

#	Current State	Event /Condition =>Action	Next State
20	WTACK	**ALPMR_Activate_req ()** => ERRCLS:= ALPMR ERRCODE:= WRONG-STATE ALPMR_Activate_cnf (-)	WTACK
21	WTACK	**ALPMR_Alarm_Ack.req ()** => ERRCLS:= ALPMR ERRCODE:= WRONG-STATE ALPMR_Alarm_Ack.cnf (-)	WTACK
22	WTACK	**ALPMR_Close_req ()** => ALPMR_Close_cnf () ALPMR_Error_ind ()	W-START
23	WTACK	**APMR_A_Data.ind ()** => ERRCLS:= RTA_ERR_CLS_PROTOCOL ERRCODE:= AR protocol violation ALPMR_Error_ind ()	WTACK
24	WTACK	**APMS_A_Data.cnf (-)** => ERRCLS:= ALPMR ERRCODE:= Invalid ALPMR_Error_ind ()	WTACK

5.6.2.5 Functions, Macros, Timers and Variables

Table 1067 contains the functions, macros, timers and variables used by the ALPMR and their arguments and their descriptions.

Table 1067 – Functions, Macros, Timers and Variables used by ALPMR

Name	Type	Meaning
AlarmInstance	Variable	This AR global variable contains the required information for this protocol machine instance.
ERRCLS	Variable	This local variable contains the entity signaling the error.
ERRCODE	Variable	This local variable contains the error reason in the context of the ERRCLS.

5.6.3 Device

5.6.3.1 General

5.6.3.1.1 General

The CMDEV arranges the connection establishment of an IO device. Figure 197 shows the integration of the IO device CM in conjunction with RPC and/or RSI.

The extensions CIM and “Connection to CIM” shall be used for ARBlockReq.ARProperties.-TimeAwareSystem == TimeAware.

The extensions CIM and “Connection to CIM” shall be used for ARBlockReq.ARProperties.-ProtectionProperties != 0.

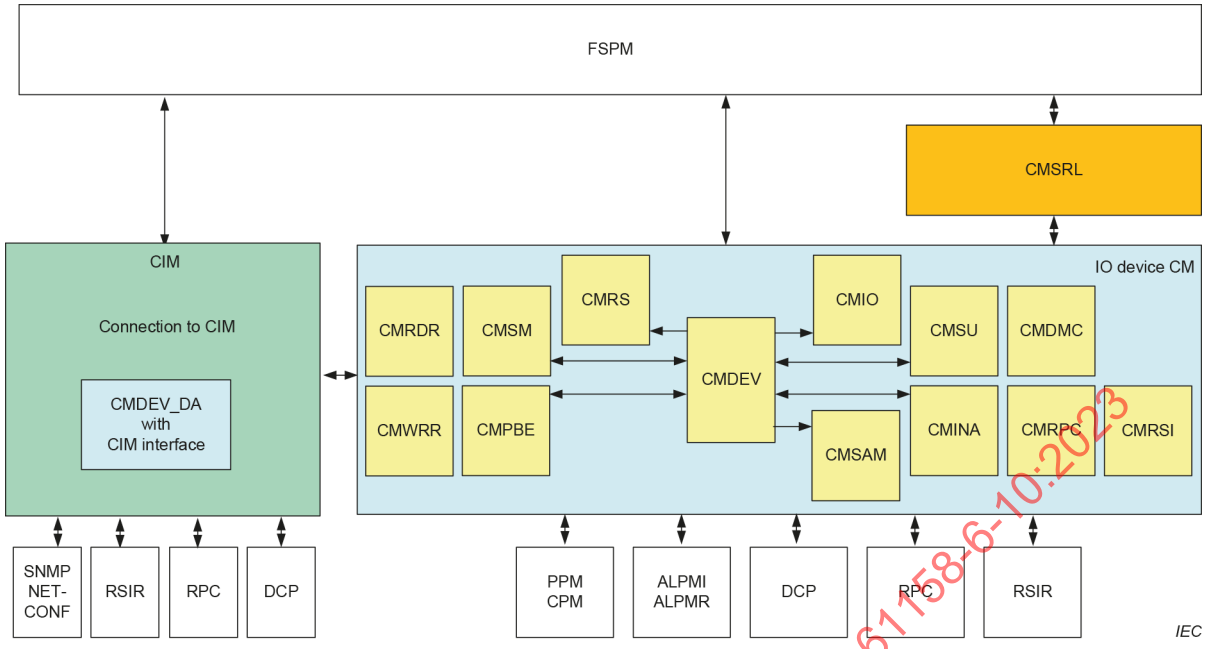


Figure 197 – Scheme of the IO device CM

CMDEV
This state machine handles the context management of the IO device.

CMWRR
This state machine handles the responder functionality of the write service.

CMRDR
This state machine handles the responder functionality of the read service.

CMSM
This state machine handles the connection monitoring during the startup.

CMPBE
This state machine handles the PrmBegin, PrmEnd and ApplRdy sequence. It is the basis for system redundancy and dynamic reconfiguration.

CMSU
This state machine handles the startup of the different state machines during startup and shutdown.

CMIO
This state machine handles the IO Data services.

CMDMC
This state machine handles the startup of the multicast communication relations.

CMRPC
This state machine handles the translation of CMDEV services to RPC services and vice versa.

CMINA
This state machine handles IP address and name availability.

CMRS
This state machine handles start and stop of the reporting system.

CMSRL
This state machine handles higher reliability based on ARsets. It is the basis for system redundancy.

CMRSI
This state machine handles the translation between RSI services and RM services.

CIM
The CIM is used to access, based on CMDEV_DA, the CIM ASE of the IO device in order to read and write the required managed objects. See 5.6.5.

5.6.3.1.2 State transition diagram

The state transition diagram of the IO device CM is shown in Figure 198.

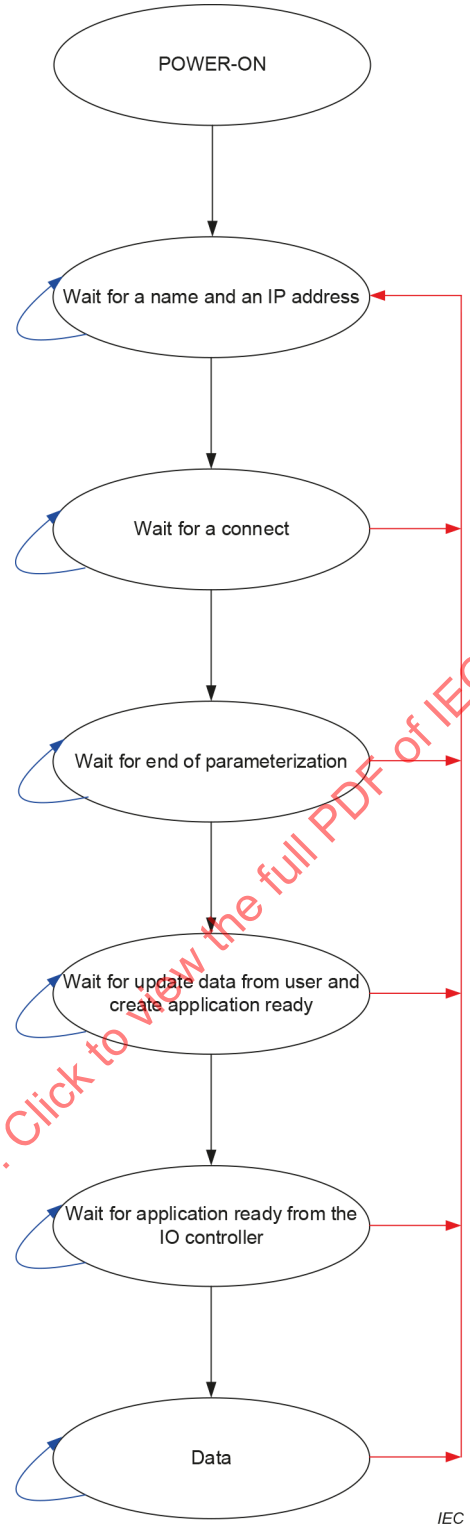


Figure 198 – State transition diagram of the IO device CM

States of the IO device CM are:

POWER-ON	Startup the IO device and initialize all necessary resources
Wait for a name and an IP address	A name and an IP address is needed for the communication to the IO controller
Wait for connect	The IO device is the responder for the application relation to the IO controller. The connect establishes an application relation between the IO controller and the IO device. Part of this AR are the required communication resources and the requested submodules.
Wait for end of parameterization	The IO controller parameterizes all submodules of the AR and informs the application of the IO device when the last startup record is conveyed. The application uses this signal to evaluate the parameters and executes them.
Wait for update from user and create application ready	The submodules generate interpretable data after the parameterization. This data shall be updated in the PPM buffer before the application ready is signaled to the IO controller.
Wait for application ready from IO controller	The IO controller has received the application ready from the IO device and executes it locally. After that, the confirmation of the application ready shall be given to the IO device.
Data	Cyclic data exchange of the submodules between the IO controller and IO devices, with the first NewData_Ind after the reception of the application ready confirmation from the IO controller. See Annex A for the special case isochronous application

5.6.3.2 Context Management Device

5.6.3.2.1 CMDEV state machine

5.6.3.2.1.1 Primitive definitions

####### 5.6.3.2.1.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Device (CMDEV) are described in the service definition and shown in Table 1068.

Table 1068 – Remote primitives issued or received by CMDEV

Primitive	Source	Destination	Associated parameters	Functions
CM_Ccontrol.req	FSPMDEV	CMDEV	AREP, ControlBlock, ModuleDiffBlock	The service Ccontrol conveys the application ready flag.
CM_Connect.rsp (-)	FSPMDEV	CMDEV	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	This service primitive is a negative response and the requested application relationship is not established.
CM_Connect.rsp (+)	FSPMDEV	CMDEV	AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock	This service primitive is a positive response to the request to establish an application relationship.
CM_Dcontrol.rsp (-)	FSPMDEV	CMDEV	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	This service primitive is a negative response to end of parameter signal.
CM_Dcontrol.rsp (+)	FSPMDEV	CMDEV	AREP, ControlBlock	This service primitive is a positive response to end of parameter signal.

Primitive	Source	Destination	Associated parameters	Functions
RM_Ccontrol.cnf (-)	CMRPC	CMDEV	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Ccontrol.cnf (+)	CMRPC	CMDEV	AREP, ControlBlock	—
RM_Connect.ind	CMRPC	CMDEV	AREP, ARBlockReq, ListOfIOCRBlockReq, AlarmCRBlockReq, ListOfExpectedSubmodule BlockReq, ListOfMCRBlockReq, ListOfSubframeBlockReq	—
RM_Dcontrol.ind	CMRPC	CMDEV	AREP, ControlBlock	—
RM_Release.ind	CMRPC	CMDEV	AREP, ControlBlock	—
CM_Ccontrol.cnf (-)	CMDEV	FSPMDEV	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	This service primitive confirms the Ccontrol service.
CM_Ccontrol.cnf (+)	CMDEV	FSPMDEV	AREP, ControlBlock	This service primitive confirms the Ccontrol service.
CM_Connect.ind	CMDEV	FSPMDEV	AREP, ARBlockReq, ListOfIOCRBlockReq, AlarmCRBlockReq, ListOfExpectedSubmodule BlockReq, ListOfMCRBlockReq, ListOfSubframeBlockReq	This service primitive indicates a request to establish an application relationship.
CM_Dcontrol.ind	CMDEV	FSPMDEV	AREP, ControlBlock	This service primitive indicates the end of parameter.
CM_Release.ind	CMDEV	FSPMDEV	AREP, ReleaseBlock	This service primitive indicates that the application relationship should be released.
RM_Ccontrol.req	CMDEV	CMRPC	AREP, ControlBlock, ModuleDiffBlock	—
RM_Connect.rsp (-)	CMDEV	CMRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2	—
RM_Connect.rsp (+)	CMDEV	CMRPC	AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock	—
RM_Dcontrol.rsp (+)	CMDEV	CMRPC	AREP, ControlBlock	—
RM_Release.rsp (+)	CMDEV	CMRPC	AREP, ControlBlock	—

5.6.3.2.1.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMDEV are described in the service definition and shown in Table 1069.

Table 1069 – Local primitives issued or received by CMDEV

Primitive	Source	Destination	Associated parameters	Functions
CMIO_info_ind	CMIO	CMDEV	—	—
CMSU_start_cnf (-)	CMSU	CMDEV	ListOfCREPs, ListOfErrors	This service primitive confirms the start service.
CMSU_start_cnf (+)	CMSU	CMDEV	ListOfCREPs	This service primitive confirms the start service.
CM_Abort_req	FSPMDEV	CMDEV	AREP	The service Abort removes the stored AREP from the list of AREPs.
CM_Init_req	FSPMDEV	CMDEV	—	—
CMSU_start_req	CMDEV	CMSU	—	This service primitive confirms the start service.
CM_Abort_cnf	CMDEV	FSPMDEV	AREP	The service Abort removes the stored AREP from the list of AREPs.
CM_Init_cnf	CMDEV	FSPMDEV	—	—
CMDEV_state_ind	CMDEV	FSPMDEV CMSU CMIO CMWRR CMSM CMPBE CMRPC	state {ABORT, STARTUP, PRMEND, APPLRDY, DATA}	This service primitive controls the state of the AR establishment.
RSI_R_Add_req ()	CMDEV	RSIRN	RSAP, VendorID, DeviceID, Instance	—
RSI_R_Add_cnf ()	RSIRN	CMDEV	RSAP, PNIOStatus	—

5.6.3.2.1.2 State transition diagram

The state transition diagram of the CMDEV is shown in Figure 199.

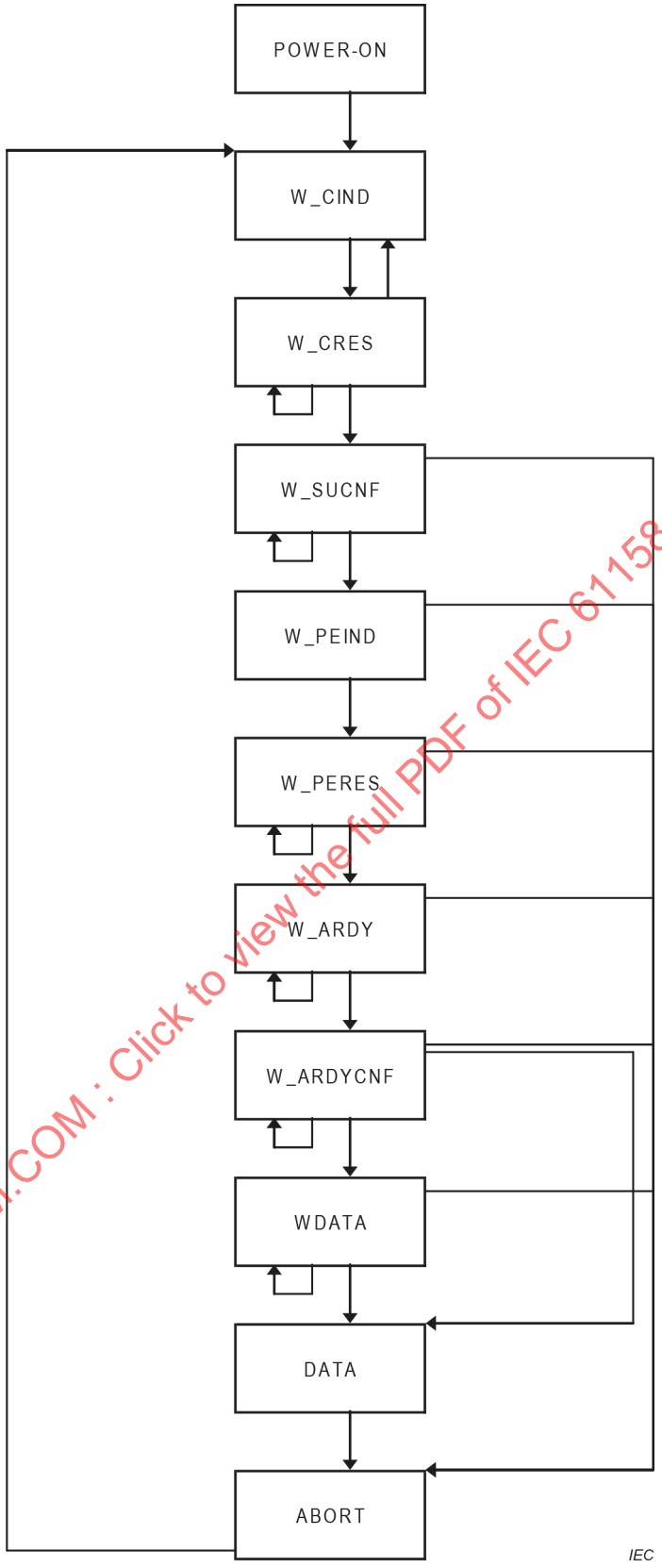


Figure 199 – State transition diagram of CMDEV

States of the CMDEV are:

POWER-ON	Data initialization
W_CIND	Wait for connect indication
W_CRES	Wait for connect response from the application and start the CMSU
W_SUCNF	Wait for CMSU confirmation and convey the connect response
W_PEIND	Wait for PrmEnd indication
W_PERES	Wait for PrmEnd response from the application
W_ARDY	Wait for ARDY request from the application
W_ARDYCNF	Wait for ARDY confirmation
WDATA	Wait for the established cyclic data exchange to start the CPM / PPM connection monitoring. The ALPMI is now able to issue an alarm.
DATA	Data exchange and connection monitoring using the CPM / PPM. The ALPMI is now able to issue an alarm.
ABORT	Abort application relation

5.6.3.2.1.3 State machine description

The CM Device protocol machine (CMDEV) is present for each AR of an IO Device.

Two kinds of ARs exist, selected by the attribute ARProperties.DeviceAccess and handled by CMDEV and CMDEV_DA.

Furthermore, if a multicast communication relation (MCR) is selected, the MCITimeoutFactor shall be used to optimize the diagnosis. If a MCR is not running and the MCITimeoutFactor has not expired, the ControlBlockConnect(ApplRdy).req shall be delayed until the MCITimeoutFactor expires or until all MCRs are running, if earlier.

The possible RPC reruns are handled at the CMRPC.

Special case: Dynamic Reconfiguration
The ACCM handling needs to consider that two ARs can be established to the same device and thus, the removal of one of these ARs shall not remove the assigned ACCM entry.

5.6.3.2.1.4 CMDEV state table

Table 1070 contains the complete description of the CMDEV state machine.

Table 1070 – CMDEV state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	**CM_Init_req ()** => RSI_R_Add_req () CM_Init_cnf ()	W_CIND
2	W_CIND	**RM_Connect.ind ()** /CheckAPDU () == OK => Ready4DATA:=FALSE CM_Connect.ind ()	W_CRES
3	W_CIND	**RM_Connect.ind ()** /CheckAPDU () == ErrorDecode, ErrorCode1, ErrorCode2 => RM_Connect.rsp (-)	W_CIND
4	W_CRES	**CM_Connect.rsp (+)** => CMSU_start_req () //NOTE Start all required protocol machines	W_SUCNF
5	W_CRES	**CM_Connect.rsp (-)** => CreateLogBookEntry () RM_Connect.rsp (-)	W_CIND
6	W_CRES	**RM_Dcontrol.ind ()** => ignore	W_CRES
7	W_CRES	**RM_Release.ind ()** => ignore	W_CRES
8	W_SUCNF	**CMSU_start_cnf (+)** => CMDEV_state_ind (STARTUP) RM_Connect.rsp (+)	W_PEIND
9	W_SUCNF	**CMSU_start_cnf (-)** => ErrorDecode := PNIO ErrorCode1 := CMDEV ErrorCode2 := state conflict CreateLogBookEntry () RM_Connect.rsp (-)	ABORT
10	W_SUCNF	**RM_Dcontrol.ind ()** => ignore	W_SUCNF
11	W_SUCNF	**RM_Release.ind ()** => ignore	W_SUCNF
12	W_PEIND	**RM_Dcontrol.ind ()** /PRMEND => CM_Dcontrol.ind ()	W_PERES
13	W_PEIND	**RM_Release.ind ()** => CM_Release.ind () RM_Release.rsp ()	ABORT
14	W_PERES	**CM_Dcontrol.rsp (+)** => CMDEV_state_ind (PRMEND) RM_Dcontrol.rsp (+)	W_ARDY
15	W_PERES	**CM_Dcontrol.rsp (-)** => RM_Dcontrol.rsp (-)	ABORT

#	Current State	Event /Condition =>Action	Next State
16	W_PERES	**RM_Dcontrol.ind ()** => ignore	W_PERES
17	W_PERES	**RM_Release.ind ()** => ignore	W_PERES
18	W_ARDY	**PdevCheckFailed ()** => ErrorDecode := PNIO ErrorCode1 := RTA_ERR_CLS_PROTOCOL ErrorCode2 := “Pdev, no port offers required speed/duplex mode” CreateLogBookEntry ()	ABORT
19	W_ARDY	**CMIO_info_ind ()** /state == DATA_POSSIBLE => Ready4DATA:=TRUE	W_ARDY
20	W_ARDY	**CMIO_info_ind ()** /state == DATA_IMPOSSIBLE => Ready4DATA:=FALSE	W_ARDY
21	W_ARDY	**CM_Ccontrol.req ()** /APPLRDY && All PPM buffer are updated and valid => CMDEV_state_ind (APPLRDY) RM_Ccontrol.req (APPLRDY)	W_ARDYCNF
22	W_ARDY	**CM_Ccontrol.req ()** /APPLRDY && ! All PPM buffer are updated and valid => CM_Ccontrol.cnf (-)	W_ARDY
23	W_ARDY	**RM_Release.ind ()** => CM_Release.ind () RM_Release.rsp ()	ABORT
24	W_ARDYCNF	**CMIO_info_ind ()** /state == DATA_POSSIBLE => Ready4DATA:=TRUE	W_ARDYCNF
25	W_ARDYCNF	**CMIO_info_ind ()** /state == DATA_IMPOSSIBLE => Ready4DATA:=FALSE	W_ARDYCNF

26	W_ARDYCNF	**RM_Ccontrol.cnf (+)**
		/Ready4DATA == TRUE
		=>
		CMDEV_state_ind (DATA)
		CM_Ccontrol.cnf (+)
		
		//NOTE 1 The next NewData_ind () after the “Local AR In Data.ind” contains the first valid data from the IO controller
		
		//NOTE 2 Alarms can be issued by the ALPMI
			DATA

27	W_ARDYCNF	**RM_Ccontrol.cnf (+)** /Ready4DATA == FALSE => CM_Ccontrol.cnf (+) //NOTE Alarms can be issued by the ALPMI	WDATA

#	Current State	Event /Condition =>Action	Next State
28	W_ARDYCNF	**RM_Ccontrol.cnf (-)** => CreateLogBookEntry () CM_Ccontrol.cnf (-)	ABORT
29	W_ARDYCNF	**RM_Release.ind ()** => CM_Release.ind () RM_Release.rsp ()	ABORT
30	WDATA	**CMIO_info_ind ()** /state == DATA_POSSIBLE => Ready4DATA:=TRUE CMDEV_state_ind (DATA)	DATA
31	WDATA	**CMIO_info_ind ()** /state == DATA_IMPOSSIBLE => Ready4DATA:=FALSE	WDATA
32	WDATA	**RM_Release.ind ()** => CM_Release.ind () RM_Release.rsp ()	ABORT
33	DATA	**RM_Release.ind ()** => CM_Release.ind () RM_Release.rsp ()	ABORT
34	ABORT	 => CMDEV_state_ind (ABORT)	W_CIND
35	ANY	**CMDEV_state_ind ()** /state == ABORT => ignore	W_CIND
36	ANY	**CM_Abort_req ()** => CMDEV_state_ind (ABORT)	W_CIND
37	ANY	**RSI_R_Add_cnf ()** => ignore	ANY

5.6.3.2.1.5 Functions, Macros, Timers and Variables

Table 1071 contains the functions, macros, timers and variables used by the CMDEV and their arguments and their descriptions.

Table 1071 – Functions, Macros, Timers and Variables used by CMDEV

Name	Type	Function/Meaning
CheckAPDU	Function	This local function checks the semantics of the connect. Special case: Reserved for security "SecurityMode.Mode == Protected" Only a protected RM_Connect.ind is accepted.
PdevCheckFailed	Function	This local function checks whether at least one port offers an operational Media Attachment Unit (MAU) with at least 100 Mbit/s speed and full duplex mode, after the parameterization is applied. Special case: “10 Mbit/s only devices” This local function checks whether at least one port offers an operational Media Attachment Unit (MAU) with at least 10 Mbit/s speed and full duplex mode, after the parameterization is applied.
All PPM buffers are updated and valid	Macro	This local macro returns TRUE if the user has updated the Data_Buffer of the PPMs.
Ready4Data	Variable	This local variable is used to store state information during AR establishment.

5.6.3.2.2 CMDEV Device Access state machine

5.6.3.2.2.1 General

The CMDEV_DA arranges the device access connection establishment of an IO device. Figure 200 shows the integration of the IO device CM – device access.

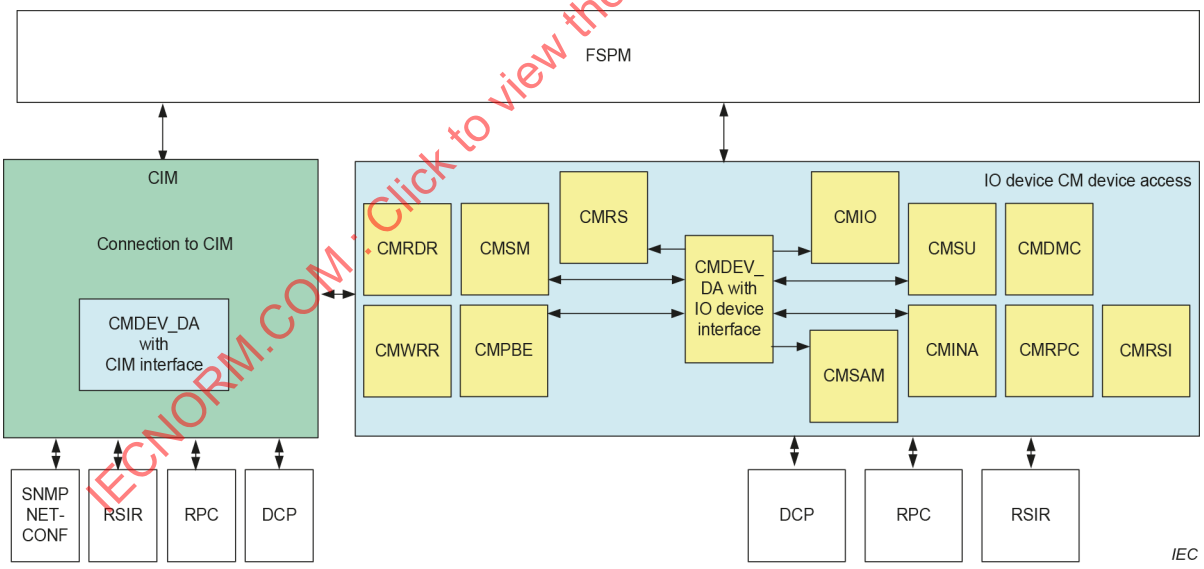


Figure 200 – Scheme of the IO device CM – device access

CMDEV_DA
This state machine handles the context management of the IO device.

CMWRR
This state machine handles the responder functionality of the write service.

CMRDR
This state machine handles the responder functionality of the read service.

CMSM
This state machine handles the connection monitoring during the startup.

CMPBE
This state machine handles the PrmBegin, PrmEnd and ApplRdy sequence. It is the basis for system redundancy and dynamic reconfiguration.

CMSU
This state machine handles the startup of the different state machines during startup and shutdown.

CMIO
This state machine handles the IO Data services.

CMDMC
This state machine handles the startup of the multicast communication relations.

CMRPC
This state machine handles the translation of CMDEV services to RPC services and vice versa.

CMINA
This state machine handles IP address and name availability.

CMRS
This state machine handles start and stop of the reporting system.

CMSRL
This state machine handles higher reliability based on ARsets. It is the basis for system redundancy.

CMRSI
This state machine handles the translation between RSI services and RM services.

CIM
The CIM is used to access, based on CMDEV_DA, the CIM ASE of the IO device in order to read and write the required managed objects. See 5.6.5.

5.6.3.2.2.2 Primitive definitions

5.6.3.2.2.2.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Device – Device Access (CMDEV_DA) are described in the service definition and shown in Table 1072.

Table 1072 – Remote primitives issued or received by CMDEV_DA

Primitive	Source	Destination	Associated parameters	Functions
CM_Connect.ind	CMDEV_DA	FSPMDEV	AREP, ARBlockReq	—
RM_Connect.rsp(-)	CMDEV_DA	CMRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2	—
RM_Connect.rsp(+)	CMDEV_DA	CMRPC	AREP, ARBlockRes	—
RM_Release.rsp(-)	CMDEV_DA	CMRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2	—
RM_Release.rsp(+)	CMDEV_DA	CMRPC	AREP, IODReleaseRes	—

Primitive	Source	Destination	Associated parameters	Functions
CM_Connect.rsp (-)	FSPMDEV	CMDEV_DA	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
CM_Connect.rsp (+)	FSPMDEV	CMDEV_DA	AREP, ARBlockRes	—
RM_Connect.ind	CMRPC	CMDEV_DA	AREP, ARBlockReq	—
RM_Release.ind	CMRPC	CMDEV_DA	AREP, IODReleaseReq	—

5.6.3.2.2.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMDEV_DA are described in the service definition and shown in Table 1073.

Table 1073 – Local primitives issued or received by CMDEV_DA

Primitive	Source	Destination	Associated parameters	Functions
CMDEV_state_ind	CMDEV_DA	FSPMDEV CMSU CMIO CMWRR CMSM CMPBE CMRPC	AREP, state {STARTUP, ABORT}	—
CM_Abort_req	FSPMDEV	CMDEV_DA	AREP	—
CM_Abort_cnf	CMDEV_DA	FSPMDEV	AREP	—
CM_Init_req	FSPMDEV	CMDEV_DA	AREP	—
CM_Init_cnf	CMDEV_DA	FSPMDEV	AREP	—

5.6.3.2.3 State transition diagram

The state transition diagram of the CMDEV_DA is shown in Figure 201.

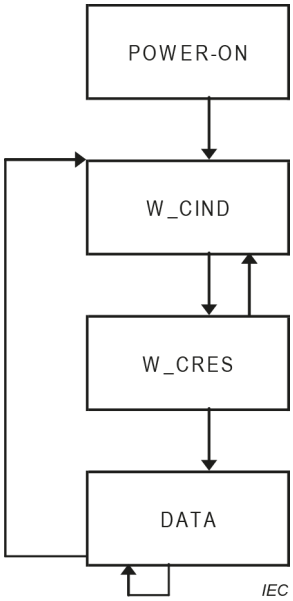


Figure 201 – State transition diagram of CMDEV_DA

States of the CMDEV_DA are:

POWER-ON	Data initialization
W_CIND	Wait for connect indication
W_CRES	Wait for connect response from the application
DATA	Data exchange monitored by CMSM

5.6.3.2.2.4 State machine description

The CM Device protocol machine for device access (CMDEV_DA) is present for each AR of an IO Device. The selection is done by the attribute ARProperties.DeviceAccess.

The device access is used if only a record service channel to the application is needed. The connection monitoring for this kind of an IOCAR is done by the CMSM using read and write record services.

The CMDEV_DA signals to the CMSM only STARTUP when switching to DATA in order to keep the monitoring active using read and write services.

5.6.3.2.2.5 CMDEV_DA (device access) state table

Table 1074 contains the complete description of the CMDEV_DA state machine.

Table 1074 – CMDEV_DA state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	**CM_Init_req ()** => CM_Init_cnf ()	W_CIND
2	W_CIND	**RM_Connect.ind ()** /CheckAPDU () == OK => CM_Connect.ind ()	W_CRES
3	W_CIND	**RM_Connect.ind ()** /CheckAPDU () == ErrorDecode, ErrorCode1, ErrorCode2 => RM_Connect.rsp (-)	W_CIND
4	W_CRES	**CM_Abort_req ()** => RM_Connect.rsp (-) CMDEV_state_ind (ABORT) CM_Abort_cnf ()	W_CIND
5	W_CRES	**CM_Connect.rsp (+)** => RM_Connect.rsp (+) CMDEV_state_ind (STARTUP)	DATA
6	W_CRES	**CM_Connect.rsp (-)** => RM_Connect.rsp (-)	W_CIND
7	DATA	**CM_Abort_req ()** => CMDEV_state_ind (ABORT) CM_Abort_cnf ()	W_CIND
8	DATA	**RM_Release.ind ()** => RM_Release.rsp () CMDEV_state_ind (ABORT)	W_CIND
9	DATA	**CM_Connect.rsp ()** => ignore	DATA

5.6.3.2.2.6 Functions, Macros, Timers and Variables

Table 1075 contains the functions, macros, timers and variables used by the CMDEV_DA and their arguments and their descriptions.

Table 1075 – Functions, Macros, Timers and Variables used by CMDEV_DA

Name	Type	Function/Meaning
CheckAPDU	Function	This local function checks the semantics of the connect.

5.6.3.3 Context Management Startup Device

5.6.3.3.1 Primitive definitions

5.6.3.3.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Startup Protocol Machine Device (CMSU) are described in the service definition and shown in Table 1076.

Table 1076 – Remote primitives issued or received by CMSU

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

5.6.3.3.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMSU are described in the service definition and shown in Table 1077.

Table 1077 – Local primitives issued or received by CMSU

Primitive	Source	Destination	Associated parameters	Functions
ACCM_req	CMSU	ACCM	Command={ADD}, IPAddress, MACAddress	—
ALPMI_Activate_req	CMSU	ALPMI	CREP	—
ALPMR_Activate_req	CMSU	ALPMR	CREP	—
APMR_Activate_req	CMSU	APMR	CREP	—
APMS_Activate_req	CMSU	APMS	CREP	—
CMSU_start_cnf (-)	CMSU	CMDEV	ErrorList {ListOfPPM, ListOfCPM, ListOfMCR, ALPMI, ALPMR, DFP}	This service primitive starts the underlying protocol resources.
CMSU_start_cnf (+)	CMSU	CMDEV	—	—
CMDMC_Activate_req	CMSU	CMDMC	CREP.CMDMC, ProviderStationName, CR_Parameter	—
CMDMC_Close_req	CMSU	CMDMC	CREP	—
ACCM_cnf	ACCM	CMSU	Command={REMOVE}, IPAddress	—
ALPMI_Activate_cnf (-)	ALPMI	CMSU	—	—
ALPMI_Error_ind	ALPMI	CMSU	CREP, ERRCLS, ERRCODE	—
ALPMR_Activate_cnf (-)	ALPMR	CMSU	—	—
ALPMR_Error_ind	ALPMR	CMSU	CREP, ERRCLS, ERRCODE	—
APMR_Activate_cnf (-)	APMR	CMSU	—	—
APMR_Error_ind	APMR	CMSU	CREP, ERRCLS, ERRCODE	—

Primitive	Source	Destination	Associated parameters	Functions
APMS_Activate_cnf (-)	APMS	CMSU	—	—
APMS_Error_ind	APMS	CMSU	CREP, ERRCLS, ERRCODE	—
CMSU_start_req	CMDEV	CMSU	AREP, ARBlockReq, ListOfIOCRBlockReq, AlarmCRBlockReq, ListOfExpectedSubmoduleBlockReq, ListOfMCRBlockReq, ListOfSubframeBlockReq, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes	This service primitive starts the underlying protocol resources.
CMDMC_Activate_cnf (-)	CMDMC	CMSU	—	—
CMDMC_Error_ind ()	CMDMC	CMSU	ErrorCode1, ErrorCode2	—
CPM_Activate_cnf (-)	CPM	CMSU	—	—
CPM_Error_ind	CPM	CMSU	ErrorCode1, ErrorCode2	—
DFP_Activate_cnf (-)	DFP	CMSU	—	—
PPM_Activate_cnf (-)	PPM	CMSU	—	—
PPM_Error_ind	PPM	CMSU	ErrorCode1, ErrorCode2	—
XXX_Activate_cnf (+)	xxx	CMSU	CREP	—
XXX_Close_cnf (+)	xxx	CMSU	CREP	—
CPM_Activate_req	CMSU	CPM	CREP, DA, SA, FrameID, RxOption, ARProperties.StartupMode, …	—
CPM_Close_req	CMSU	CPM	CREP	—
DFP_Activate_req	CMSU	DFP	CREP	—
DFP_Close_req	CMSU	DFP	CREP	—
CMDEV_state_ind	CMDEV CMSU	FSPMDEV CMSU CMIO CMWRR CMSM CMPBE CMRPC	State {ABORT, STARTUP, PRMEND, APPLRDY, DATA}	This service primitive controls the state of the CMSU.
PPM_Activate_req	CMSU	PPM	CREP, DA, SA, FrameID, TxOption, ARProperties.StartupMode, …	—
PPM_Close_req	CMSU	PPM	CREP	—

5.6.3.3.2 State transition diagram

The state transition diagram of the CMSU is shown in Figure 202.

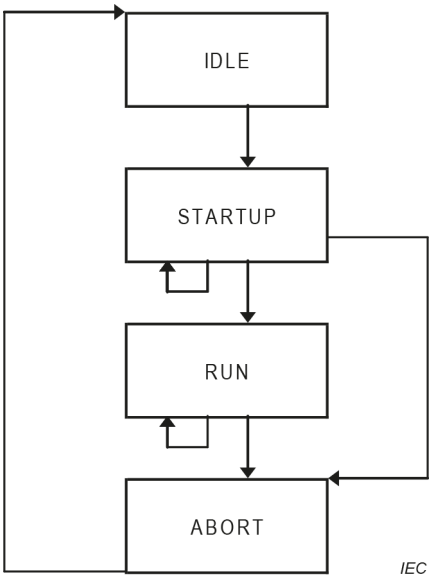


Figure 202 – State transition diagram of CMSU

States of the CMSU are:

IDLE	Wait for the start command and start the associated state machines.
STARTUP	Check the response of the started machines and continue in state RUN.
RUN	Startup of the state machines is done, wait for shutdown or fault.
ABORT	Termination sequence

5.6.3.3.3 State machine description

The Context Management Startup Protocol Machine Device (CMSU) of an IO device arranges the start of the underlying protocol machines.

The startup of the CPM and PPM state machines shall be done between the reception of the connect request service and of the Application ready service. The startup shall be finished before issuing the Application ready request service.

5.6.3.3.4 CMSU state table

Table 1078 contains the description of the CMSU state machine.

Table 1078 – CMSU state table

#	Current State	Event /Condition =>Action	Next State

1	IDLE	**CMSU_start_req ()**
		=>
		CREP.PPM := create (PPM)
		CREP.CPM := create (CPM)
		
		if exist:
		CREP.MPPM := create (PPM)
		CREP.CMDMC := create (CMDMC)
		CREP.DFP := create (DFP)
		
		//NOTE Create high and low alarm machine instances
		CREP.ALPMI := create (ALPMI)
		CREP.ALPMR := create (ALPMR)
		CREP.APMS := create (APMS)
		CREP.APMR := create (APMR)
		
		PPM_Activate_req ()
		CPM_Activate_req ()
		
		if applicable
		CMDMC_Activate_req ()
		if applicable
		DFP_Activate_req ()
		
		ALPMI_Activate_req ()
		ALPMR_Activate_req ()
		APMS_Activate_req ()
		APMR_Activate_req ()
		
		// Create static ARP cache entry
		ACCM_req (Command:=ADD, IPAddress, MACAddress)
			STARTUP

2	STARTUP	**XXX_Activate_cnf (+)** /if last confirmation is received => CMSU_start_cnf (+)	STARTUP
3	STARTUP	**PPM_Activate_cnf (-)** => ErrorCode2 = AR add provider or consumer failed CMSU_start_cnf (-)	STARTUP
4	STARTUP	**CPM_Activate_cnf (-)** => ErrorCode2 = AR add provider or consumer failed CMSU_start_cnf (-)	STARTUP
5	STARTUP	**ALPMI_Activate_cnf (-)** => ErrorCode2 = AR alarm-open failed CMSU_start_cnf (-)	STARTUP
6	STARTUP	**ALPMR_Activate_cnf (-)** => ErrorCode2 = AR alarm-open failed CMSU_start_cnf (-)	STARTUP
7	STARTUP	**APMS_Activate_cnf (-)** => ErrorCode2 = AR alarm-open failed CMSU_start_cnf (-)	STARTUP
8	STARTUP	**APMR_Activate_cnf (-)** => ErrorCode2 = AR alarm-open failed CMSU_start_cnf (-)	STARTUP
9	STARTUP	**CMDMC_Activate_cnf (-)** => ErrorCode1 := CMDEV ErrorCode2 := state conflict CMSU_start_cnf (-)	STARTUP

#	Current State	Event /Condition =>Action	Next State
10	STARTUP	**DFP_Activate_cnf (-)** => ErrorCode2 = AR add provider or consumer failed CMSU_start_cnf (-)	STARTUP
11	STARTUP	**CMSU_start_req ()** => ErrorCode1 := CMDEV ErrorCode2 := state conflict CMSU_start_cnf (-)	STARTUP
12	STARTUP	**CMDEV_state_ind ()** /state != ABORT => ignore	RUN

13	STARTUP	**CMDEV_state_ind ()**
		/state == ABORT
		=>
		PPM_Close_req ()
		CPM_Close_req ()
		
		if applicable
		  CMDMC_Close_req ()
		if applicable
		  DFP_Close_req ()
		
		ALPMI_Close_req ()
		ALPMR_Close_req ()
		APMS_Close_req ()
		APMR_Close_req ()
		
		// Cleanup static ARP cache
		ACCM_req (Command:=REMOVE, IPAddress)
			ABORT

14	RUN	**CMDEV_state_ind ()** /state != ABORT => ignore	RUN

15	RUN	**CMDEV_state_ind ()**
		/state == ABORT
		=>
		PPM_Close_req ()
		CPM_Close_req ()
		
		if applicable
		  CMDMC_Close_req ()
		if applicable
		  DFP_Close_req ()
		
		ALPMI_Close_req ()
		ALPMR_Close_req ()
		APMS_Close_req ()
		APMR_Close_req ()
		
		// Cleanup static ARP cache
		ACCM_req (Command:=REMOVE, IPAddress)
			ABORT

16	RUN	**CMSU_start_req ()** => ErrorCode1 := CMDEV ErrorCode2 := state conflict CMSU_start_cnf (-)	RUN

#	Current State	Event /Condition =>Action	Next State

17	ABORT	**XXX_Close_cnf (+)**
		/if last confirmation is received
		=>
		Delete (PPM)
		Delete (CPM)
		
		Delete (ALPMI)
		Delete (ALPMR)
		
		Delete (APMS)
		Delete (APMR)
		
		if exist:
		Delete (DFP)
		
		if exist:
		Delete (CMDMC)
		
		//NOTE Delete all created PPMs and CPMs
			IDLE

18	ANY	**PPM_Error_ind ()** => CMDEV_state_ind (ABORT)	SAME
19	ANY	**CPM_Error_ind ()** => CMDEV_state_ind (ABORT)	SAME
20	ANY	**ALPMI_Error_ind ()** => ErrorCode2 := AR alarm-send or ErrorCode2 := AR alarm-ack-send CMDEV_state_ind (ABORT)	SAME
21	ANY	**ALPMR_Error_ind ()** => ErrorCode2 := AR alarm-ind CMDEV_state_ind (ABORT)	SAME
22	ANY	**APMS_Error_ind ()** => ErrorCode2 := AR alarm-send or ErrorCode2 := AR alarm-ack-send CMDEV_state_ind (ABORT)	SAME
23	ANY	**APMR_Error_ind ()** => ErrorCode2 := AR alarm-ind CMDEV_state_ind (ABORT)	SAME
24	ANY	**CMDMC_Error_ind ()** => CMDEV_state_ind (ABORT)	SAME
25	ANY	**ACCM_cnf ()** => ignore	SAME

5.6.3.3.5 Functions, Macros, Timers and Functions

Table 1079 contains the functions, macros, timers and variables used by the CMSU and their arguments and their descriptions.

Table 1079 – Functions, Macros, Timers and Variables used by the CMSU

Name	Type	Function/Meaning
Delete	Function	This local function frees or unbounds the machines from the CMCTL instance if the AR is aborted.
If last confirmation is received	Macro	This local macro collects all the confirmations and returns TRUE if all are received with xxx_cnf (+).

5.6.3.4 Context Management Input Output Device

5.6.3.4.1 Primitive definitions

5.6.3.4.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Input Output Protocol Machine Device (CMIO) are described in the service definition and shown in Table 1080.

Table 1080 – Remote primitives issued or received by CMIO

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

5.6.3.4.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by Context Management Input Output Protocol Machine Device (CMIO) are described in the service definition and shown in Table 1081.

Table 1081 – Local primitives issued or received by CMIO

Primitive	Source	Destination	Associated parameters	Functions
CMIO_info_ind	CMIO	CMDEV CMSM	info	—
CPM_State_ind	CPM	CMIO	—	—
CPM_NewData_ind	CPM	CMIO FSPMDEV CMDMC CTLIO	AREP, CREP, APDU_Status, data {Data, NoData}	This service primitive indicates an update of a receive buffer.
CMDEV_state_ind	CMDEV	FSPMDEV CMSU CMIO CMWRR CMSM CMPBE CMRPC	State {ABORT, STARTUP, PRMEND, APPLRDY}	This service primitive controls the state of the CMIO.

5.6.3.4.2 State transition diagram

The state transition diagram of the CMIO is shown in Figure 203.

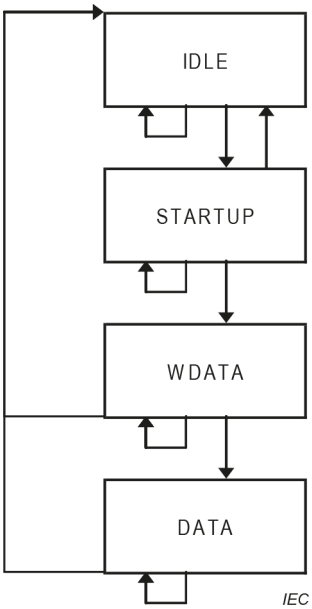


Figure 203 – State transition diagram of CMIO

States of the CMIO are:

IDLE	No AR context exists
STARTUP	Initial parameterization phase
WDATA	Wait until the AR is ready to switch to the DATA state, because all associated CPMs are receiving frames.
DATA	Data exchange phase. Monitoring the DataHoldTime.

5.6.3.4.3 State machine description

The Context Management Input Output Protocol Machine Device (CMIO) arranges the access to the DataStatus and NewData indication.

5.6.3.4.4 CMIO state table

Table 1082 contains the complete description of the CMIO state machine.

Table 1082 – CMIO state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**CPM_NewData_ind ()** => ignore	IDLE
2	IDLE	**CPM_State_ind ()** => ignore	IDLE
3	IDLE	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
4	IDLE	**CMDEV_state_ind ()** /state == STARTUP => For all CPMs CmInstance[CREP.CPM]:=Stop	STARTUP
5	STARTUP	**CPM_NewData_ind ()** => ignore	STARTUP
6	STARTUP	**CPM_State_ind ()** => ignore	STARTUP
7	STARTUP	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
8	STARTUP	**CMDEV_state_ind ()** /state == PRMEND => StartTimer ()	WDATA
9	WDATA	**CPM_NewData_ind ()** => CmInstance[CREP.CPM]:=Start	WDATA
10	WDATA	**CPM_State_ind ()** /state == STOP => CmInstance[CREP.CPM]:=Stop	WDATA
11	WDATA	**CPM_State_ind ()** /state == START => CmInstance[CREP.CPM]:=Start	WDATA
12	WDATA	**TimerExpired ()** /All CPM of CmInstance[CREP.CPM] contain the value “Start” => StartTimer () CMIO_info_ind (DATA_POSSIBLE)	WDATA
13	WDATA	**TimerExpired ()** /One CPM of CmInstance[CREP.CPM] contains the value “Stop” => StartTimer () CMIO_info_ind (DATA_IMPOSSIBLE)	WDATA
14	WDATA	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
15	WDATA	**CMDEV_state_ind ()** /state == APPLRDY => ignore	WDATA

#	Current State	Event /Condition =>Action	Next State
16	WDATA	**CMDEV_state_ind ()** /state == DATA => StopTimer ()	DATA
17	DATA	**CPM_NewData_ind ()** => ignore	DATA
18	DATA	**CPM_State_ind ()** /state == STOP && CREP != CREP.MCPM => CMDEV_state_ind (ABORT)	DATA
19	DATA	**CPM_State_ind ()** /state == START => ignore	DATA
20	DATA	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE

5.6.3.4.5 Functions, Macros, Timers and Variables

Table 1083 contains the functions, macros, timers and variables used by the CMIO and their arguments and their descriptions.

Table 1083 – Functions used by the CMIO

Name	Type	Function/Meaning
StartTimer	Function	This local function starts or restarts a timer.
StopTimer	Function	This local function stops a timer.
TimerExpired	Function	This local function is issued if a timer expires.
CmInstance	Variable	This AR global variable contains the information about the AR.
IntervalTime	Timer	This local timer creates an event for the checking of the local states. The recommended time for this timer is 100 ms. NOTE An implementation can use “change” events instead of this timer.

5.6.3.5 Context Management Reporting System

5.6.3.5.1 Primitive definitions

5.6.3.5.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Reporting System Protocol Machine Device (CMRS) are described in the service definition and shown in Table 1084.

Table 1084 – Remote primitives issued or received by CMRS

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

5.6.3.5.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMRS are described in the service definition and shown in Table 1085.

Table 1085 – Local primitives issued or received by CMRS

Primitive	Source	Destination	Associated parameters	Functions
CMDEV_state_ind	CMDEV	FSPMDEV CMSU CMIO CMWRR CMSM CMPBE CMRPC CMRS	state {ABORT, STARTUP, PRMEND, APPLRDY, DATA}	This service primitive controls the state of the AR establishment.

5.6.3.5.2 State transition diagram

The state transition diagram of the CMRS is shown in Figure 204.

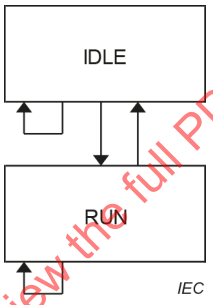


Figure 204 – State transition diagram of CMRS

States of the CMRS are:

IDLE Wait for the start command and start the associated state machines.

RUN Startup of the state machines is done, wait for shutdown or fault.

5.6.3.5.3 State machine description

The Context Management Reporting System Protocol Machine Device (CMRS) of an IO device arranges the start and stop of the Reporting System.

The startup of the Reporting System state machine shall be done after the reception of the Parameterization end service. The startup shall be finished before issuing the Application ready request service.

The CMRS shall be stopped when the AR is aborted.

5.6.3.5.4 CMRS state table

Table 1086 contains the description of the CMRS state machine.

Table 1086 – CMRS state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**CMDEV_state_ind ()** /state != PrmEnd || RSInfoBlock != TRUE => ignore	IDLE
2	IDLE	**CMDEV_state_ind ()** /state == PrmEnd && RSInfoBlock == TRUE => InitializeReportingSystem ()	RUN
3	RUN	**CMDEV_state_ind ()** /state != Abort => ignore	RUN
4	RUN	**CMDEV_state_ind ()** /state == Abort => DeleteReportingSystem ()	IDLE

5.6.3.5.5 Functions, Macros, Timers and Functions

Table 1087 contains the functions, macros, timers and variables used by the CMRS and their arguments and their descriptions.

Table 1087 – Functions, Macros, Timers and Variables used by the CMRS

Name	Type	Function/Meaning
InitializeReportingSystem	Function	This local function allocates and initializes the buffers, state flags and sequence numbers.
DeleteReportingSystem	Function	This local function frees the resources.
RSInfoBlock	Variable	This local variable is set to: TRUE, if the Connect service contains an RSInfoBlock FALSE, if the Connect service does not contain an RSInfoBlock.

5.6.3.6 Context Management Write Record Device

5.6.3.6.1 Primitive definitions

5.6.3.6.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Write Record Responder Protocol Machine Device (CMWRR) are described in the service definition and shown in Table 1088.

Table 1088 – Remote primitives issued or received by CMWRR

Primitive	Source	Destination	Associated parameters	Functions
CM_Write.rsp (-)	FSPMDEV	CMWRR	AREP, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
CM_Write.rsp (+)	FSPMDEV	CMWRR	AREP, SeqNumber, AddData1, AddData2	—
RM_Write.ind	CMRPC	CMWRR	AREP, API, SlotNumber, SubslotNumber, Index, SeqNumber, Length, Data	—
CM_Write.ind	CMWRR	FSPMDEV	AREP, API, SlotNumber, SubslotNumber, Index, PrmFlag, SeqNumber, Length, Data	—
RM_Write.rsp (-)	CMWRR	CMRPC	AREP, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Write.rsp (+)	CMWRR	CMRPC	AREP, SeqNumber	—

5.6.3.6.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMWRR are described in the service definition and shown in Table 1089.

Table 1089 – Local primitives issued or received by CMWRR

Primitive	Source	Destination	Associated parameters	Functions
CMDEV_state_ind	CMDEV	FSPMDEV CMSU CMIO CMWRR CMSM CMPBE CMRPC	state {ABORT, STARTUP, PRMEND, APPLRDY, DATA}	—
CMSRL_state_ind	CMSRL	CMWRR	state {PRIMARY, BACKUP}	This service primitive controls the state of an AR of an ARset.

5.6.3.6.2 State transition diagram

The state transition diagram of the CMWRR is shown in Figure 205.

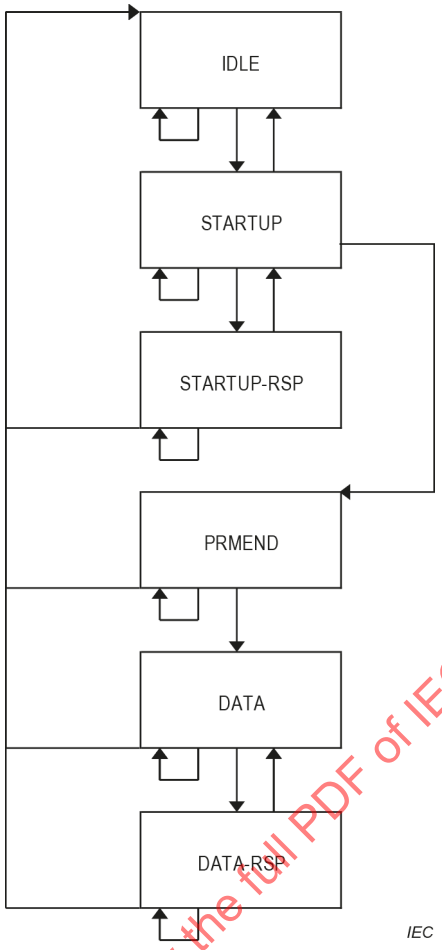


Figure 205 – State transition diagram of CMWRR

States of the CMWRR are:

IDLE	No AR context exists
STARTUP	The associated AR reaches the internal state STARTUP. Write services are handled during this state.
STARTUP-RSP	Wait for the user response and convey it to the requester.
PRMEND	The associated AR reaches the internal state PRMEND. Write services are rejected during this state.
DATA	The associated AR reaches the internal state DATA. Write services are handled during this state.
DATA-RSP	Wait for the user response and convey it to the requester.

5.6.3.6.3 State machine description

The CM Write Record protocol machine (CMWRR) arranges the write record service.

5.6.3.6.4 CMWRR state table

Table 1090 contains the description of the CMWRR state machine. The CMWRR checks whether the ARUUID exists.

Special case: MultipleWrite in ARstate “BACKUP”
The MultipleWrite.req is handled as a packet containing records which are checked one by one

by CMWRR. The MultipleWrite.rsp contains the response created by CMWRR per record "Write.rsp (-) Backup" and itself responds "MultipleWrite.rsp (+) OK".

Special case: CoC in ARstate “BACKUP”
The CombinedObjectContainer Write.req is handled as a transparent package like a single record and responds "Write.rsp (-) Backup" for the whole package.

Table 1090 – CMWRR state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**RM_Write.ind ()** => ErrorDecode := PNIORW ErrorCode1 := state conflict RM_Write.rsp (-)	IDLE
2	IDLE	**CM_Write.rsp ()** => ignore	IDLE
3	IDLE	**CMDEV_state_ind ()** /state == STARTUP => ignore	STARTUP
4	IDLE	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
5	IDLE	**CMSRL_state_ind ()** => ignore	IDLE
6	STARTUP	**CMDEV_state_ind ()** /state == PRMEND => ignore	PRMEND
7	STARTUP	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
8	STARTUP	**CMSRL_state_ind ()** => ignore	STARTUP
9	STARTUP	**RM_Write.ind ()** /ARstate != Backup => CM_Write.ind ()	STARTUP-RSP
10	STARTUP	**RM_Write.ind ()** /ARstate == Backup => ErrorDecode := PNIORW ErrorCode1 := backup RM_Write.rsp (-)	STARTUP
11	STARTUP-RSP	**CM_Write.rsp (+)** => RM_Write.rsp (+)	STARTUP
12	STARTUP-RSP	**CM_Write.rsp (-)** => RM_Write.rsp (-)	STARTUP
13	STARTUP-RSP	**RM_Write.ind ()** => ignore	STARTUP-RSP

#	Current State	Event /Condition =>Action	Next State
14	STARTUP-RSP	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
15	STARTUP-RSP	**CMSRL_state_ind ()** => ignore	STARTUP-RSP
16	PRMEND	**RM_Write.ind ()** => ErrorDecode := PNIORW ErrorCode1 := state conflict RM_Write.rsp (-)	PRMEND
17	PRMEND	**CMDEV_state_ind ()** /state == APPLRDY => ignore	DATA
18	PRMEND	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
19	PRMEND	**CMSRL_state_ind ()** => ignore	PRMEND
20	DATA	**CMDEV_state_ind ()** /state == DATA => ignore	DATA
21	DATA	**RM_Write.ind ()** /ARstate != Backup => CM_Write.ind ()	DATA-RSP
22	DATA	**RM_Write.ind ()** /ARstate == Backup => ErrorDecode := PNIORW ErrorCode1 := backup RM_Write.rsp (-)	DATA
23	DATA	**CM_Write.rsp (+)** => ignore	DATA
24	DATA	**CM_Write.rsp (-)** => ignore	DATA
25	DATA	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
26	DATA	**CMSRL_state_ind ()** => ignore	DATA
27	DATA-RSP	**CM_Write.rsp (+)** => RM_Write.rsp (+)	DATA
28	DATA-RSP	**CM_Write.rsp (-)** => RM_Write.rsp (-)	DATA
29	DATA-RSP	**RM_Write.ind ()** => ignore	DATA-RSP

#	Current State	Event /Condition =>Action	Next State
30	DATA-RSP	**CMSRL_state_ind ()** /ARstate == Backup => ErrorDecode := PNIORW ErrorCode1 := backup RM_Write.rsp (-)	DATA
31	DATA-RSP	**CMSRL_state_ind ()** /ARstate != Backup => ignore	DATA-RSP
32	DATA-RSP	**CMDEV_state_ind ()** /state == DATA => ignore	DATA-RSP
33	DATA-RSP	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE

5.6.3.6.5 Functions, Macros, Timers and Variables

Table 1091 contains the functions, macros, timers and variables used by the CMWRR and their arguments and their descriptions.

Table 1091 – Functions, Macros, Timers and Variables used by CMWRR

Name	Type	Function/Meaning
ARstate	Variable	This AR global variable contains the current state of the AR. Possible values are: First, Primary, Backup Initial value of this variable is Primary

5.6.3.7 Context Management Read Record Device

5.6.3.7.1 Primitive definitions

5.6.3.7.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Read Record Responder Protocol Machine Device (CMRDR) are described in the service definition and shown in Table 1092.

Table 1092 – Remote primitives issued or received by CMRDR

Primitive	Source	Destination	Associated parameters	Functions
CM_Read.ind	CMRDR	FSPMDEV	AREP, API, TargetUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length	—
CM_Read.rsp (+)	FSPMDEV	CMRDR	AREP, SeqNumber, Length, Data	—
CM_Read.rsp (-)	FSPMDEV	CMRDR	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Read.ind	CMRPC	CMRDR	AREP, API, TargetUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length, Data	—
RM_Read.rsp (+)	CMRDR	CMRPC	AREP, SeqNumber, Length, Data	—
RM_Read.rsp (-)	CMRDR	CMRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
CM_ReadQuery.ind	CMRDR	FSPMDEV	AREP, API, TargetUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length, Data	—
CM_ReadQuery.rsp (+)	FSPMDEV	CMRDR	AREP, SeqNumber, Length, Data	—
CM_ReadQuery.rsp (-)	FSPMDEV	CMRDR	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—

5.6.3.7.1.2 Primitives exchanged between local machines

The local primitives including their associated parameters issued or received by CMRDR are described in the service definition and shown in Table 1093.

Table 1093 – Local primitives issued or received by CMRDR

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

5.6.3.7.2 State transition diagram

The state transition diagram of the CMRDR is shown in Figure 206.

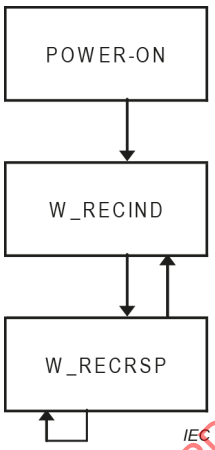


Figure 206 – State transition diagram of CMRDR

States of the CMRDR are:

POWER-ON	Data initialization
W_RECIND	Wait for read record indication
W_RECRSP	Wait for read record response

5.6.3.7.3 State machine description

The CM Read Record protocol machine (CMRDR) arranges the read record service.

5.6.3.7.4 CMRDR state table

Table 1094 contains the complete description of the CMRDR state machine. The CMRPC checks whether the ARUUID exists.

Table 1094 – CMRDR state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> Init	W_RECIND
2	W_RECIND	**RM_Read.ind ()** /Length == 0 => CM_Read.ind ()	W_RECRSP
3	W_RECIND	**RM_Read.ind ()** /Length > 0 => CM_ReadQuery.ind ()	W_RECRSP
4	W_RECIND	**CM_Read.rsp ()** => ignore	W_RECIND
5	W_RECIND	**CM_ReadQuery.rsp ()** => ignore	W_RECIND
6	W_RECRSP	**CM_Read.rsp (+)** => RM_Read.rsp (+)	W_RECIND
7	W_RECRSP	**CM_Read.rsp (-)** => RM_Read.rsp (-)	W_RECIND
8	W_RECRSP	**CM_ReadQuery.rsp (+)** => RM_Read.rsp (+)	W_RECIND
9	W_RECRSP	**CM_ReadQuery.rsp (-)** => RM_Read.rsp (-)	W_RECIND
10	W_RECRSP	**RM_Read.ind ()** => ignore	W_RECRSP

The serialization of the “Read” RPC calls for one AR is done by the RPC layer or the RSI layer. Thus, it is not necessary to check the serialization again in the CMRDR.

5.6.3.7.5 Functions, Macros, Timers and Variables

Table 1095 contains the functions, macros, timers and variables used by the CMRDR and their arguments and their descriptions.

Table 1095 – Functions, Macros, Timers and Variables used by CMRDR

Name	Type	Function/Meaning
—	—	—

5.6.3.8 Context Management Surveillance Device

5.6.3.8.1 Primitive definitions

5.6.3.8.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Surveillance Protocol Machine Device (CMSM) are described in the service definition and shown in Table 1096.

Table 1096 – Remote primitives issued or received by CMSM

Primitive	Source	Destination	Associated parameters	Functions
CM_Read.ind	CMRDR	CMSM	AREP, API, TargetUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length	—
CM_Read.rsp	CMRDR	CMSM	AREP, SeqNumber	—
CM_Write.ind	CMWRR	CMSM	AREP, API, SlotNumber, SubslotNumber, Index, PrmFlag, SeqNumber, Length, Data	—
CM_Write.rsp	CMWRR	CMSM	AREP, SeqNumber	—
RM_Read.ind	CMRPC	CMSM	AREP, API, TargetUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length	Trigger record indication This service primitive is the indication of the Read service.
RM_Read.rsp (+)	CMSM	CMRPC	AREP, API, TargetUUID, SeqNumber, Length, Data	Trigger record response This service primitive is the response to the Read service.

5.6.3.8.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMSM are described in the service definition and shown in Table 1097.

Table 1097 – Local primitives issued or received by CMSM

Primitive	Source	Destination	Associated parameters	Functions
CMDEV_state_ind	CMDEV	FSPMDEV CMSU CMIO CMWRR CMSM CMPBE CMRPC	state	State "STARTUP" starts CMI timeout State "DATA" stops CMI timeout and switches to DHT monitoring State "ABORT" terminates the AR

5.6.3.8.2 State transition diagram

The state transition diagram of the CMSM is shown in Figure 207.

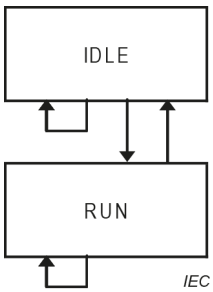


Figure 207 – State transition diagram of CMSM

States of the CMSM are:

IDLE Wait for the start of the connection monitoring

RUN Wait for the switchover to DHT monitoring and switch off the CMI monitoring

5.6.3.8.3 State machine description

The CM server protocol machine (CMSM) arranges the connection monitoring between the connect response and the start of the PPM / CPM (DHT) connection monitoring.

The CMSM uses Read, Write and Dcontrol indication and response for the triggering of the connection monitoring.

5.6.3.8.4 CMSM state table

Table 1098 contains the complete description of the CMSM state machine.

Table 1098 – CMSM state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**CMDEV_state_ind ()** /state == STARTUP => StartTimer (CMI, CMInitiatorActivityTimeout)	RUN
2	IDLE	**CMDEV_state_ind ()** /state != STARTUP => ignore	IDLE
3	IDLE	**TimerExpired (CMI)** => ignore	IDLE
4	IDLE	**RM_Read.ind ()** /Index == Trigger => RM_Read.rsp (+)	IDLE
5	IDLE	**RM_Read.ind ()** /Index != Trigger => ignore	IDLE
6	IDLE	**CM_Read.ind ()** => ignore	IDLE
7	IDLE	**CM_Read.rsp ()** => ignore	IDLE
8	IDLE	**CM_Write.ind ()** => ignore	IDLE
9	IDLE	**CM_Write.rsp ()** => ignore	IDLE
10	RUN	**TimerExpired (CMI)** => CMDEV_state_ind (ABORT)	IDLE
11	RUN	**CMDEV_state_ind ()** /state == DATA || state == ABORT => StopTimer (CMI)	IDLE
12	RUN	**CMDEV_state_ind ()** /state != DATA && state != ABORT => ignore	RUN
13	RUN	**RM_Read.ind ()** /Index == Trigger => StopTimer (CMI) StartTimer (CMI, CMInitiatorActivityTimeout) RM_Read.rsp (+)	RUN
14	RUN	**CM_Read.rsp ()** => StartTimer (CMI, CMInitiatorActivityTimeout)	RUN
15	RUN	**CM_Write.rsp ()** => StartTimer (CMI, CMInitiatorActivityTimeout)	RUN

#	Current State	Event /Condition =>Action	Next State
16	RUN	**CM_Read.ind ()** => StopTimer (CMI)	RUN
17	RUN	**CM_Write.ind ()** => StopTimer (CMI)	RUN
18	RUN	**CM_Dcontrol.rsp ()** => StartTimer (CMI, CMInitiatorActivityTimeout)	RUN
19	RUN	**CM_Dcontrol.ind ()** => StopTimer (CMI)	RUN

5.6.3.8.5 Functions, Macros, Timers and Variables

Table 1099 contains the functions, macros, timers and variables used by the CMSM and their arguments and their descriptions.

Table 1099 – Functions, Macros, Timers and Variables used by the CMSM

Name	Type	Function/Meaning
StartTimer	Function	This local function starts or restarts a timer.
StopTimer	Function	This local function stops a timer.
TimerExpired	Function	This local function is issued if a timer expires.
Trigger	Macro	This local Macro represents the Trigger record.
CMI	Timer	This local timer shall be loaded with the CMInitiatorActivityTimeout and is used for the connection monitoring during startup.

5.6.3.9 Context Management Prm Begin End Device

5.6.3.9.1 Primitive definitions

5.6.3.9.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Prm Begin End Protocol Machine Device (CMPBE) are described in the service definition and shown in Table 1100.

Table 1100 – Remote primitives received by CMPBE

Primitive	Source	Destination	Associated parameters	Functions
CM_Ccontrol.req	FSPMDEV	CMPBE	AREP, ControlBlock, ModuleDiffBlock	—
CM_Dcontrol.rsp	FSPMDEV	CMPBE	AREP, ControlBlock	—
RM_Ccontrol.cnf	CMRPC	CMPBE	AREP, ControlBlock	—
RM_Dcontrol.ind	CMRPC	CMPBE	AREP, ControlBlock	—
CM_Ccontrol.cnf	CMPBE	FSPMDEV	AREP, ControlBlock	—
CM_Dcontrol.ind	CMPBE	FSPMDEV	AREP, ControlBlock	—
RM_Ccontrol.req	CMPBE	CMRPC	AREP, ControlBlock, ModuleDiffBlock	—
RM_Dcontrol.rsp (-)	CMPBE	CMRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Dcontrol.rsp (+)	CMPBE	CMRPC	AREP, ControlBlock	—

5.6.3.9.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMPBE are described in the service definition and shown in Table 1101.

Table 1101 – Local primitives issued or received by CMPBE

Primitive	Source	Destination	Associated parameters	Functions
CMDEV_state_ind	CMDEV	FSPMDEV CMSU CMIO CMWRR CMSM CMPBE CMRPC	state	—
CM_Abort_req	CMPBE	FSPMDEV CMSU CMIO CMSM	AREP	—

5.6.3.9.2 State transition diagram

The state transition diagram of the CMPBE is shown in Figure 208.

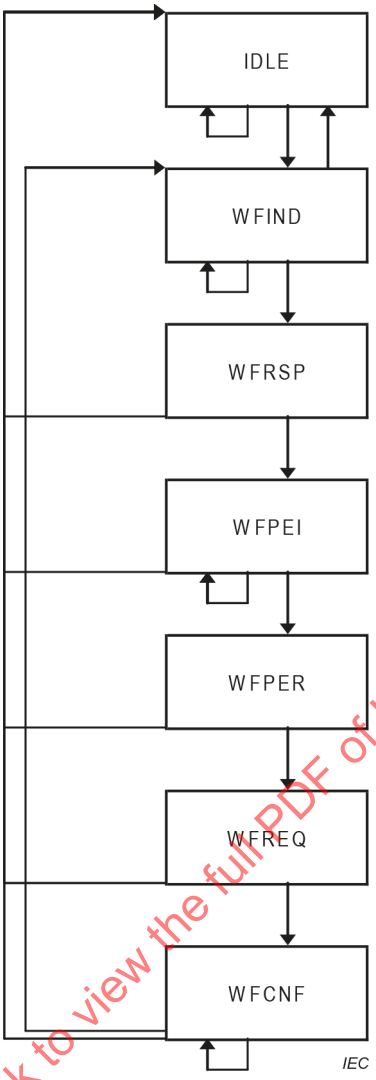


Figure 208 – State transition diagram of CMPBE

States of the CMPBE are:

IDLE	Wait for the start of DATA
WFIND	Wait for a PrmBegin command
WFRSP	Wait for a PrmBegin response from the application
WFPEI	Wait for the records from the IO controller and the PrmEnd request
WFPER	Wait for a PrmEnd response from the application
WFREQ	Wait for an update of the IO Data of the affected submodules before issuing the ApplRdy service. Wait for the application ready request from the application
WFCNF	Wait for an application ready confirmation and after that, for the next CPM_NewData_ind from the affected Submodules / IOCR for valid data.

5.6.3.9.3 State machine description

The PrmBegin PrmEnd ApplRdy sequence is used to perform a consistent update of the parameters of submodules using the well known ApplRdy model. The IO device handles this sequence using the CMPBE.

The IO device stores all alarms from the involved submodules during the PrmBegin PrmEnd ApplRdy sequence.

Any overlapping alarm, except Plug or Release, shall be proceeded and acknowledged by the IO controller before the IO device issues the ApplRdy. Overlapping Plug and Release Alarms shall be stored by the IO controller and processed after the PrmBegin PrmEnd ApplRdy sequence.

The IO device evaluates all stored Alarms from the involved submodules and removes all dispensable Alarms before issuing ApplRdy.

5.6.3.9.4 CMPBE state table

Table 1102 contains the complete description of the CMPBE state machine.

Table 1102 – CMPBE state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**CMDEV_state_ind ()** /state == DATA => ignore	WFIND
2	IDLE	**CMDEV_state_ind ()** /state =! DATA => ignore	IDLE
3	WFIND	**RM_Dcontrol.ind ()** /Stored request exists => Retrieve stored request CM_Dcontrol.ind ()	WFRSP
4	WFIND	**RM_Dcontrol.ind ()** /ControlBlock == PrmBegin => AlarmProceedingBlocked //NOTE Do not proceed any Alarm of an involved submodule CM_Dcontrol.ind ()	WFRSP
5	WFIND	**RM_Dcontrol.ind ()** /ControlBlock =! PrmBegin => ignore	WFIND
6	WFIND	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
7	WFRSP	**CM_Dcontrol.rsp ()** /ControlBlock == PrmBegin => RM_Dcontrol.rsp (+)	WFPEI
8	WFRSP	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
9	WFPEI	**RM_Dcontrol.ind ()** /ControlBlock == PrmEnd => CM_Dcontrol.ind ()	WFPER
10	WFPEI	**RM_Dcontrol.ind ()** /ControlBlock =! PrmEnd => ignore	WFPEI
11	WFPEI	**CMDEV_state_ind ()** /state == ABORT	IDLE

#	Current State	Event /Condition =>Action	Next State
		=> ignore	
12	WFPER	**CM_Dcontrol.rsp ()** /ControlBlock == PrmEnd => RM_Dcontrol.rsp (+)	WFREQ
13	WFPER	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
14	WFREQ	**CM_Ccontrol.req ()** /ControlBlock == ApplRdy && AlarmAckPending == TRUE => CM_Ccontrol.cnf (-)	WFREQ
15	WFREQ	**CM_Ccontrol.req ()** /ControlBlock == ApplRdy && AlarmAckPending == FALSE => RM_Ccontrol.req ()	WFCNF
16	WFREQ	**RM_Dcontrol.ind ()** /ControlBlock == PrmBegin //NOTE PrmEnd is handled by CMRPC as RPC Rerun => RM_Dcontrol.rsp (-) CMDEV_state_ind (ABORT)	IDLE
17	WFREQ	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE
18	WFCNF	**RM_Ccontrol.cnf ()** /ControlBlock == ApplRdy => AlarmProceedingAllowed //NOTE Continue with the proceeding of Alarms CM_Ccontrol.cnf ()	WFIND
19	WFCNF	**RM_Dcontrol.ind ()** /ControlBlock == PrmBegin => Store request //NOTE Execute the stored request after ApplRdy.cnf to make the PBE sequence robust	WFCNF
20	WFCNF	**CMDEV_state_ind ()** /state == ABORT => ignore	IDLE

5.6.3.9.5 Functions, Macros, Timers and Variables

Table 1103 contains the functions, macros, timers and variables used by the CMPBE and their arguments and their descriptions.

Table 1103 – Functions, Macros, Timers and Variables used by the CMPBE

Name	Type	Function/Meaning
AlarmAckPending	Macro	This local macro checks whether an involved submodule (list given by the PrmBegin.req) has an outstanding Alarm acknowledge for any Alarm except Plug or Release. TRUE := an outstanding Alarm acknowledge exists FALSE := no outstanding Alarm acknowledge exists
AlarmProceedingAllowed	Macro	This local macro continues the proceeding of all kinds of alarms for any involved submodule (list given by the PrmBegin).
AlarmProceedingBlocked	Macro	This local macro stops the proceeding of all kinds of alarms for any involved submodule (list given by the PrmBegin).
Retrieve stored request	Macro	This local macro retrieves the stored request.
Store request	Macro	This local macro stores the request for later use.
Stored request exists	Macro	This local macro checks whether a request is stored.

5.6.3.10 Context Management Discovery Multicast Communication

5.6.3.10.1 Primitive definitions

5.6.3.10.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management Discovery Multicast Communication Protocol Machine (CMDMC) are described in the service definition and shown in Table 1104.

Table 1104 – Remote primitives issued or received by CMDMC

Primitive	Source	Destination	Associated parameters	Functions
DCP_Identify.cnf (-)	DCPMCS	CMDMC	CREP, ERRCLS, ERRCODE	This service primitive indicates that the address resolution of an M-Provider station name failed.
DCP_Identify.cnf (+)	DCPMCS	CMDMC	CREP	This service primitive indicates that the address resolution of an M-Provider station name succeeded.
DCP_Identify.req	CMDMC	DCPMCS	CREP, DA, ListOfFilter, ResponseDelay	The service Identify starts the address resolution of the station name of an M-Provider

5.6.3.10.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMDMC are described in the service definition and shown in Table 1105.

Table 1105 – Local primitives issued or received by CMDMC

Primitive	Source	Destination	Associated parameters	Functions
CMDMC_Activate_req	CMSU	CMDMC	CREP, ProviderStationName, CR_Parameter	The service Activate initializes the CMDMC state machine.
CMDMC_Close_req	CMSU	CMDMC	CREP	The service Close deinitializes the CMDMC state machine.
CPM_Activate_cnf (-)	CPM	CMDMC	CREP, ERRCLS, ERRCODE	This service primitive indicates that the Activate service failed.
CPM_Activate_cnf (+)	CPM	CMDMC	CREP	This service primitive indicates that the Activate service succeeded.
CPM_Close_cnf	CPM	CMDMC	CREP	This service primitive indicates that the Close service succeeded.
CPM_NewData_ind	CPM	CMDMC FSPMDEV FSPMCTL CMIO CTLIO	AREP, CREP, APDU_Status, data {Data, NoData}	This service primitive indicates that new Multicast Consumer Data have been received.
CPM_State_ind	CPM	CMDMC FSPMDEV FSPMCTL CMIO CTLIO	CREP	This service primitive indicates that Data Hold Time has expired.
CMDMC_Error_ind ()	CMDMC	CMSU	CREP, ERRCLS, ERRCODE	This service primitive indicates that a problem exists.
CMDMC_Activate_cnf (-)	CMDMC	CMSU	CREP, ERRCLS, ERRCODE	This service primitive indicates that the Activate service failed.
CMDMC_Activate_cnf (+)	CMDMC	CMSU	CREP	This service primitive indicates that the Activate service succeeded.
CMDMC_Close_cnf	CMDMC	CMSU	CREP	This service primitive indicates that the Close service succeeded.
CPM_Close_req	CMSU	CPM	CREP	The service Activate deinitializes the CPM and stops the transmission of data. The schedule is cleared.
CPM_Activate_req	CMDMC	CPM	CREP, DA, SA, FrameId, Prio, VLAN, Exp_Length, DataHoldTime, Default_Value, Default_Status	The service Activate initializes the CPM and loads the schedule to the DMPM.

5.6.3.10.2 State transition diagram

The state transition diagram of the CMDMC is shown in Figure 209.

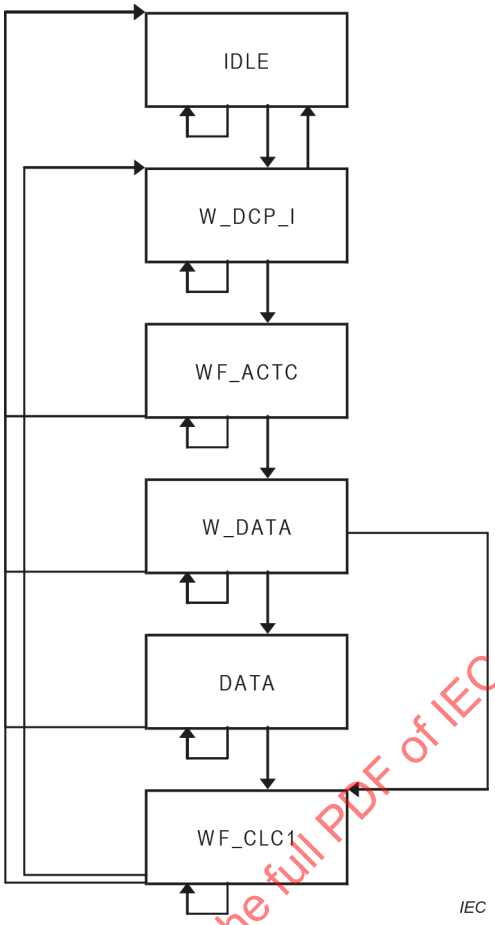


Figure 209 – State transition diagram of CMDMC

States of the CMDMC are:

IDLE	Wait for the start command
W_DCP_I	Wait for the DCP Identify service response
WF_ACTC	Wait for the activation of the CPM
W_DATA	Wait for the NewData_ind of the CPM
DATA	Working state
WF_CLC1	Wait for the shutdown of formerly started state machines

5.6.3.10.3 State machine description

The Discovery Multicast Communication Protocol Machine (CMDMC) arranges the multicast communication between IO devices. This is done by discovering the communication partners and starting of the associated CPMs. The discovery uses the DCP Identify service.

5.6.3.10.4 CMDMC state table

Table 1106 contains the complete description of the CMDMC state machine.

Table 1106 – CMDMC state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**CMDMC_Activate_req ()** => MCName := ProviderStationName DA := DCPMC_add ListOfFilter := DeviceProperties, NameOfStation, MCName ResponseDelayFactor := 1 For all Submodules within M-Consumer-CRs: Add entry to the diagnosis ASE: Submodule ChannelNumber := 0x8000 ChannelProperties.Type := 0x00 ChannelProperties.Specifier := Appears ChannelErrorType := Multicast CR Mismatch ExtChannelErrorType := Multicast Consumer CR timed out CMDMC_Activate_cnf (+) DCP_Identify.req ()	W_DCP_I
2	IDLE	**CMDMC_Close_req ()** => CMDMC_Close_cnf ()	IDLE
3	IDLE	**OTHERS** => ignore	IDLE
4	W_DCP_I	**DCP_Identify.cnf ()** /No device found => DA := DCPMC_add ListOfFilter := DeviceProperties, NameOfStation, MCName ResponseDelayFactor := 1 For all Submodules within M-Consumer-CRs: Add entry to the diagnosis ASE: ChannelNumber := 0x8000 ChannelProperties.Type := 0x00 ChannelProperties.Specifier := Appears ChannelErrorType := Multicast CR Mismatch ExtChannelErrorType := AddressResolutionFailed DCP_Identify.req () DiagnosisEvent ()	W_DCP_I
5	W_DCP_I	**DCP_Identify.cnf ()** /Device found => CREP := create(CPM) MC_InData(CREP) := FALSE CPM_Activate_req ()	WF_ACTC
6	W_DCP_I	**CMDMC_Close_req ()** => CMDMC_Close_cnf ()	IDLE
7	W_DCP_I	**OTHERS** => ignore	W_DCP_I
8	WF_ACTC	**CPM_Activate_cnf (+)** => ignore	W_DATA
9	WF_ACTC	**CPM_Activate_cnf (-)** => CMDMC_Error_ind () CPM_Close_req ()	IDLE
10	WF_ACTC	**CMDMC_Close_req ()** => CMDMC_Close_cnf ()	IDLE

#	Current State	Event /Condition =>Action	Next State
11	WF_ACTC	**OTHERS** => ignore	WF_ACTC
12	W_DATA	**CPM_NewData_ind ()** /CMDEV_ApplRdy == FALSE => MC_InData(CREP) := TRUE For all Submodules within M-Consumer-CRs: Delete M-Consumer “Multicast CR Mismatch” from the diagnosis ASE	W_DATA
13	W_DATA	**CPM_NewData_ind ()** /CMDEV_ApplRdy == TRUE => MC_InData(CREP) := TRUE For all Submodules within M-Consumer-CRs: Delete M-Consumer “Multicast CR Mismatch” from the diagnosis ASE	DATA
14	W_DATA	**CPM_State_ind ()** /state==stop => MC_InData(CREP) := FALSE For all Submodules within M-Consumer-CRs: Add entry to the diagnosis ASE: Submodule ChannelNumber := 0x8000 ChannelProperties.Type := 0x00 ChannelProperties.Specifier := Appears ChannelErrorType := Multicast CR Mismatch ExtChannelErrorType := Multicast Consumer CR timed out DiagnosisEvent () CPM_Close_req ()	WF_CLC1
15	W_DATA	**CPM_State_ind ()** /state!=stop => ignore	W_DATA
16	W_DATA	**CMDMC_Close_req ()** => CMDMC_Close_cnf ()	IDLE
17	W_DATA	**OTHERS** => ignore	W_DATA
18	DATA	**CPM_State_ind ()** /state==stop => MC_InData(CREP) := FALSE For all Submodules within M-Consumer-CRs: Add entry to the diagnosis ASE: Submodule ChannelNumber := 0x8000 ChannelProperties.Type := 0x00 ChannelProperties.Specifier := Appears ChannelErrorType := Multicast CR Mismatch ExtChannelErrorType := Multicast Consumer CR timed out DiagnosisEvent () CPM_Close_req ()	WF_CLC1
19	DATA	**CPM_State_ind ()** /state!=stop => ignore	DATA
20	DATA	**CMDMC_Close_req ()** => CMDMC_Close_cnf ()	IDLE

#	Current State	Event /Condition =>Action	Next State
21	DATA	**CPM_NewData_ind ()** => ignore	DATA
22	DATA	**OTHERS** => ignore	DATA
23	WF_CLC1	**CPM_Close_cnf ()** => DCP_Identify.req ()	W_DCP_I
24	WF_CLC1	**CMDMC_Close_req ()** => CMDMC_Close_cnf ()	IDLE
25	WF_CLC1	**OTHERS** => ignore	WF_CLC1

5.6.3.10.5 Functions, Macros, Timers and Variables

Table 1107 contains the functions, macros, timers and variables used by the CMDMC and their arguments and their descriptions.

Table 1107 – Functions, Macros, Timers and Variables used by the CMDMC

Name	Type	Function/Meaning
CMDEV_ApplRdy	Variable	This local variable indicates whether the AR is in state DATA or not
MC_InData	Variable	This local variable stores the information whether the CPM is in state RUN or not
DiagnosisEvent	Function	This local function issues an entry to the diagnosis ASE
OTHERS	Macro	This local macro contains all other possible events, not shown explicitly, in this state.

5.6.3.11 Context Management IP and Name Availability Device

5.6.3.11.1 Primitive definitions

5.6.3.11.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management IP and Name Availability Device protocol machine (CMINA) are described in the service definition and shown in Table 1108.

Table 1108 – Remote primitives issued or received by CMINA

Primitive	Source	Destination	Associated parameters	Functions
DCP_HELLO.req	CMINA	DCPHMCS	CREP, ListOfData	—
DCP_HELLO.cnf	DCPHMCS	CMINA	—	—

5.6.3.11.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMINA are described in the service definition and shown in Table 1109.

Table 1109 – Local primitives issued or received by CMINA

Primitive	Source	Destination	Associated parameters	Functions
CMDEV_state_ind	CMINA	CM*	—	Cancelation information for the AR to release resources
CIM_DatabaseUpdate_ind	CIM	CMINA	—	Informs the CMINA about an update of the CIM ASE

5.6.3.11.2 State transition diagram

The state transition diagram of the CMINA is shown in Figure 210.

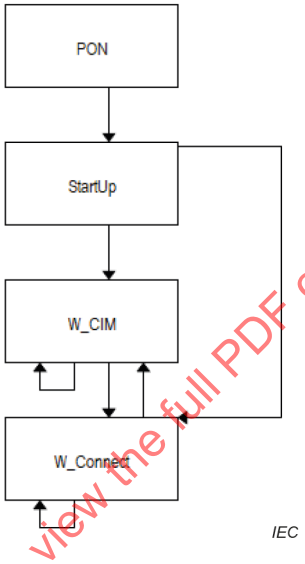


Figure 210 – State transition diagram of CMINA

States of the CMINA are:

PON	Load and apply configuration
StartUp	Wait for NameOfStation and IP-suite and start the discovery according to the stored values of the CIM database
W_CIM	Wait for a NameOfStation and optionally wait for an IP-suite
W_CONNECT	AR may be established now; changing of address parameters in CIM database is limited during an active AR

5.6.3.11.3 State machine description

The Context Management IP and Name Availability Device protocol machine (CMINA) loads and applies the configuration, and checks the condition for the acceptance of an AR establishment. It supports the DCP HELLO services for a faster device discovery by an IO Controller.

The FSHelloDelay shall be handled independently from this state machine.

A started connection establishment should stop issuing of DCP_Hello requests.

5.6.3.11.4 CMINA state table

Table 1110 contains the complete description of the CMINA state machine.

Table 1110 – CMINA state table

#	Current State	Event /Condition =>Action	Next State
1	PON	=> LoadAndExecuteConfiguration ()	StartUp
2	StartUp	/LocalNameOfStation == NIL || (IpsuiteRequiredForStartup () == TRUE && LocalIPsuite == NIL) => ignore	W_CIM
3	StartUp	/LocalNameOfStation != NIL && IpsuiteRequiredForStartup () == FALSE => if (LocalHELLOenable == TRUE) StartTimer (HelloIntervalTime) HelloCount := FSHelloRetry DCP_HELLO.req ()	W_Connect
4	StartUp	/LocalNameOfStation != NIL && IpsuiteRequiredForStartup () == TRUE && LocalIPsuite != NIL => if (LocalHELLOenable == TRUE) StartTimer (HelloIntervalTime) HelloCount := FSHelloRetry DCP_HELLO.req ()	W_Connect
5	W_CIM	**CIM_DatabaseUpdate_ind** /LocalNameOfStation == NIL || (IpsuiteRequiredForStartup () == TRUE && LocalIPsuite == NIL) => ignore	W_CIM
6	W_CIM	**CIM_DatabaseUpdate_ind** /LocalNameOfStation != NIL && IpsuiteRequiredForStartup () == FALSE => ignore	W_Connect
7	W_CIM	**CIM_DatabaseUpdate_ind** /LocalNameOfStation != NIL && IpsuiteRequiredForStartup () == TRUE && LocalIPsuite != NIL => ignore	W_Connect
8	W_CIM	**DCP_Hello.cnf ()** => ignore	W_CIM

#	Current State	Event /Condition =>Action	Next State
9	W_Connect	**CIM_DatabaseUpdate_ind** /CheckDatabase () == ChangeName || CheckDatabase () == ResetName => For all active ARs CMDEV_state_ind (ABORT)	W_CIM
10	W_Connect	**CIM_DatabaseUpdate_ind** /CheckDatabase () == ChangeIP || CheckDatabase () == ResetIP || CheckDatabase () == ChangedNMEDatabase => For all active ARs If IpsuiteRequiredForStartup () == TRUE CMDEV_state_ind (ABORT)	W_CIM
11	W_Connect	**TimerExpired (HelloIntervalTime)** => HelloCount := HelloCount – 1 If (HelloCount >= 0) StartTimer (HelloIntervalTime) DCP_HELLO.req ()	W_Connect
12	W_Connect	**DCP_Hello.cnf ()** => ignore	W_Connect

5.6.3.11.5 Functions, Macros, Timers and Variables

Table 1111 and Table 1112 contain the functions, macros, timers and variables used by the CMINA and their arguments and their descriptions.

Table 1111 – Functions, Macros, Timers and Variables used by the CMINA

Name	Type	Function/Meaning
StartTimer	Function	This local function is used to start or restart a timer.
StopTimer	Function	This local function is used to stop a timer.
TimerExpired	Function	This local function is issued if a timer expires.
LoadAndExecuteConfiguration	Function	This local function checks the CIM database (CIM ASE) for an existing configuration and applies it.
IpsuiteRequiredForStartup	Function	This local function signals, whether the node needs an IP-suite for startup.
LocalIPsuite	Macro	This local macro requests the IP-suite from the CIM ASE and returns NIL if it does not exist. It also returns NIL if the Ipstack is not available.
LocalNameOfStation	Macro	This local macro requests the NameOfStation from the CIM ASE and returns NIL if it does not exist.
LocalHELLOenable	Macro	This local macro requests the HELLO state from the CIM ASE and returns NIL if it does not exist.
HelloIntervalTime	Timer	This timer controls the interval between two HELLO requests given by the FSHelloInterval.
HelloCount	Variable	This local variable counts down the conveyed Hello request from the initial value FSHelloRetry.
CheckDatabase	Function	This local function checks the CIM database (CIM ASE) for changes of IP-suite or NameOfStation.

Table 1112 – Return values of CheckDatabase

Name	Function/Meaning
ChangeIP	ChangeIP states that the database contains an IP-suite and a different IP-suite is already stored.
ChangeName	ChangeName states that the database contains a NameOfStation and a different NameOfStation is already stored. If an AR exists, abort reason "DCP – station-name changed" shall be used.
ChangedNMEDatabase	ChangedNMEDatabase states that the database contains a changed or deleted NMEParameterUUID. If an AR exists, abort reason "NME – no or wrong configuration" shall be used.
ResetIP	ResetIP states that the database contains a delete IP-suite.
ResetName	ResetName states that the database contains a delete NameOfStation. If an AR exists, abort reason "DCP – station-name changed" shall be used.

5.6.3.12 Context Management RPC Device

5.6.3.12.1 Primitive definitions

5.6.3.12.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management RPC Device Protocol Machine (CMRPC) are described in the service definition and shown in Table 1113.

Table 1113 – Remote primitives issued or received by CMRPC

Primitive	Source	Destination	Associated parameters	Functions
RM_Ccontrol.req	CMDEV	CMRPC	AREP, ControlBlock, ModuleDiffBlock	—
RM_Connect.rsp (-)	CMDEV	CMRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Connect.rsp (+)	CMDEV	CMRPC	AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock	—
RM_Dcontrol.rsp (-)	CMDEV	CMRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Dcontrol.rsp (+)	CMDEV	CMRPC	AREP, ControlBlock	—
RM_Read.rsp (-)	CMSM, CMRDR	CMRPC	AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Read.rsp (+)	CMSM, CMRDR	CMRPC	AREP, SeqNumber, AddData1, AddData2	—

Primitive	Source	Destination	Associated parameters	Functions
RM_Release.rsp (-)	CMDEV	CMRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Release.rsp (+)	CMDEV	CMRPC	AREP, ControlBlock	—
RM_Write.rsp (-)	CMWRR	CMRPC	AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Write.rsp (+)	CMWRR	CMRPC	AREP, SeqNumber, AddData1, AddData2	—
RM_Ccontrol.cnf (-)	CMRPC	CMDEV	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Ccontrol.cnf (+)	CMRPC	CMDEV, CMPBE	AREP, ControlBlock	—
RM_Connect.ind	CMRPC	CMDEV, CMPBE	AREP, ARBlockReq, ListOfIOCRBlockReq, AlarmCRBlockReq, ListOfExpectedSubmoduleBlockReq	—
RM_Dcontrol.ind	CMRPC	CMDEV, CMDEV_DA	AREP, ControlBlock	—
RM_Read.ind	CMRPC	CMSM, CMRDR	AREP, API, TargetUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length	—
RM_Release.ind	CMRPC	CMDEV, CMDEV_DA	AREP, ControlBlock	—
RM_Write.ind	CMRPC	CMWRR	AREP, API, SlotNumber, SubslotNumber, Index, Multiple, SeqNumber, Length, Data	—
RPC_Ccontrol.req	CMRPC	RPC	Arg	—
RPC_Connect.rsp (-)	CMRPC	RPC	Arg	—
RPC_Connect.rsp (+)	CMRPC	RPC	Arg	—
RPC_Dcontrol.rsp (-)	CMRPC	RPC	Arg	—
RPC_Dcontrol.rsp (+)	CMRPC	RPC	Arg	—
RPC_Read.rsp (-)	CMRPC	RPC	Arg	—
RPC_Read.rsp (+)	CMRPC	RPC	Arg	—

Primitive	Source	Destination	Associated parameters	Functions
RPC_Release.rsp (-)	CMRPC	RPC	Arg	—
RPC_Release.rsp (+)	CMRPC	RPC	Arg	—
RPC_Write.rsp (-)	CMRPC	RPC	Arg	—
RPC_Write.rsp (+)	CMRPC	RPC	Arg	—
RPC_Ccontrol.cnf (-)	RPC	CMRPC	Arg	—
RPC_Ccontrol.cnf (+)	RPC	CMRPC	Arg	—
RPC_Connect.ind	RPC	CMRPC	Arg	—
RPC_Dcontrol.ind	RPC	CMRPC	Arg	—
RPC_Read.ind	RPC	CMRPC	Arg	—
RPC_Release.ind	RPC	CMRPC	Arg	—
RPC_Write.ind	RPC	CMRPC	Arg	—
RPC_IOXSecure.ind	RPC	CMRPC	Arg	—
RPC_IOXSecure.rsp	CMRPC	RPC	Arg	—

5.6.3.12.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMRPC are described in the service definition and shown in Table 1114.

Table 1114 – Local primitives issued or received by CMRPC

Primitive	Source	Destination	Associated parameters	Functions
CMDEV_state_ind	CMDEV	FSPMDEV CMSU CMIO CMWRR CMSM CMPBE CMRPC	AREP, state	Cancelation information for the AR to release resources
RM_IOXSecure_ind	CMRPC	CMSAM	AREP, Arg	—
RM_IOXSecure_rsp	CMSAM	CMRPC	AREP, PNIOStatus, Arg	—

5.6.3.12.2 State transition diagram

The state transition diagram of the CMRPC is shown in Figure 211.

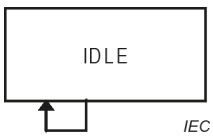


Figure 211 – State transition diagram of CMRPC

State of the CMRPC is:

IDLE Wait for an RPC call, check its integrity, the resources and for idempotent rerun

5.6.3.12.3 State machine description

The Context Management RPC Device Protocol Machine (CMRPC) arranges the RPC communication and handles all issued and received RPC services.

For devices which support two entries for UUID_IO_DeviceInterface in the EPM but allow only one to be used at a given time, the following behavior is intended:

– Remove second unused entry as soon as one is used and add the removed one again when the first one is no longer used, or

– Block second unused entry as soon as one is used and respond with e.g. "NCA_s_fault_context_mismatch"

5.6.3.12.4 CMRPC state table

Table 1115 contains the complete description of the CMRPC state machine.

Table 1115 – CMRPC state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**RM_Ccontrol.req ()** => encode RPC PDU RPC_Ccontrol.req ()	IDLE
2	IDLE	**RPC_Ccontrol.cnf (-)** /CheckRPC () == Valid => AREP := LocateAR () RM_Ccontrol.cnf (-)	IDLE
3	IDLE	**RPC_Ccontrol.cnf (+)** /CheckRPC () == Valid => AREP := LocateAR () RM_Ccontrol.cnf (+)	IDLE
4	IDLE	**RPC_Ccontrol.cnf ()** /CheckRPC () == InValid, ErrorDecode, ErrorCode1, ErrorCode2 => AREP := LocateAR () RM_Ccontrol.cnf (-)	IDLE
5	IDLE	**RPC_Dcontrol.ind ()** /CheckRPC () == Rerun => Retrieve stored response encode RPC PDU RPC_Dcontrol.rsp ()	IDLE
6	IDLE	**RPC_Dcontrol.ind ()** /CheckRPC () == Valid, ExplicitAR => AREP := LocateAR () RM_Dcontrol.ind ()	IDLE
7	IDLE	**RPC_Dcontrol.ind ()** /CheckRPC () == Valid, UnknownAR => ErrorCode2 := ARUUID_Unknown encode RPC Error PDU RPC_Dcontrol.rsp (-)	IDLE
8	IDLE	**RPC_Dcontrol.ind ()** /CheckRPC () == InValid, ErrorDecode, ErrorCode1, ErrorCode2 => encode RPC Error PDU RPC_Dcontrol.rsp (-)	IDLE

#	Current State	Event /Condition =>Action	Next State
9	IDLE	**RM_Dcontrol.rsp (+)** => encode RPC PDU Store response RPC_Dcontrol.rsp (+)	IDLE
10	IDLE	**RM_Dcontrol.rsp (-)** => encode Error PDU Store response RPC_Dcontrol.rsp (-)	IDLE
11	IDLE	**RPC_Write.ind ()** /CheckRPC () == Valid, ExplicitAR => AREP := LocateAR () RM_Write.ind ()	IDLE
12	IDLE	**RPC_Write.ind ()** /CheckRPC () == Valid, UnknownAR => ErrorCode2 := ARUUID_Unknown encode Error PDU RPC_Write.rsp (-)	IDLE
13	IDLE	**RPC_Write.ind ()** /CheckRPC () == InValid, ErrorDecode, ErrorCode1, ErrorCode2 => encode RPC Error PDU RPC_Write.rsp (-)	IDLE
14	IDLE	**RM_Write.rsp (+)** => encode RPC PDU RPC_Write.rsp (+)	IDLE
15	IDLE	**RM_Write.rsp (-)** => encode Error PDU RPC_Write.rsp (-)	IDLE
16	IDLE	**RPC_Read.ind ()** /CheckRPC () == Valid, ImplicitAR => AREP := NIL RM_Read.ind ()	IDLE
17	IDLE	**RPC_Read.ind ()** /CheckRPC () == Valid, ExplicitAR => AREP := LocateAR () RM_Read.ind ()	IDLE
18	IDLE	**RPC_Read.ind ()** /CheckRPC () == Valid, UnknownAR => ErrorCode2 := ARUUID_Unknown encode Error PDU RPC_Read.rsp (-)	IDLE
19	IDLE	**RPC_Read.ind ()** /CheckRPC () == InValid, ErrorDecode, ErrorCode1, ErrorCode2 => encode RPC Error PDU RPC_Read.rsp (-)	IDLE
20	IDLE	**RM_Read.rsp (+)** => encode RPC PDU RPC_Read.rsp (+)	IDLE
21	IDLE	**RM_Read.rsp (-)** => encode Error PDU RPC_Read.rsp (-)	IDLE

#	Current State	Event /Condition =>Action	Next State
22	IDLE	**RPC_Connect.ind ()** /CheckRPC () == Valid, UnknownAR && CheckResource () == Unavailable => ErrorCode2 := NO_AR_RESOURCE encode RPC Error PDU RPC_Connect.rsp (-)	IDLE
23	IDLE	**RPC_Connect.ind ()** /CheckRPC () == Valid, UnknownAR, UnknownARset && CheckResource () == Available => AllocateResource () ARstate (FIRST) RM_Connect.ind ()	IDLE
24	IDLE	**RPC_Connect.ind ()** /CheckRPC () == Valid, UnknownAR, ExplicitARset //NOTE Preallocated resources by the first AR of an ARset => ARstate (BACKUP) RM_Connect.ind ()	IDLE
25	IDLE	**RPC_Connect.ind ()** /CheckRPC () == Valid, UnknownAR, NoARset && CheckResource () == Available => AllocateResource () ARstate (NONE) RM_Connect.ind ()	IDLE
26	IDLE	**RPC_Connect.ind ()** /CheckRPC () == Valid, ExplicitAR => ErrorCode2 := StateConflict encode RPC Error PDU RPC_Connect.rsp (-) CMDEV_state_ind (ABORT)	IDLE
27	IDLE	**RPC_Connect.ind ()** /CheckRPC () == InValid, ErrorDecode, ErrorCode1, ErrorCode2 => encode RPC Error PDU RPC_Connect.rsp (-)	IDLE
28	IDLE	**RM_Connect.rsp (+)** => encode RPC PDU RPC_Connect.rsp (+)	IDLE
29	IDLE	**RM_Connect.rsp (-)** => FreeResource () encode Error PDU RPC_Connect.rsp (-)	IDLE
30	IDLE	**RPC_Release.ind ()** /CheckRPC () == Valid, ExplicitAR => AREP := LocateAR () RM_Release.ind ()	IDLE
31	IDLE	**RPC_Release.ind ()** /CheckRPC () == Valid, UnknownAR => ErrorCode2 := ARUUID_Unknown encode RPC Error PDU RPC_Release.rsp (-)	IDLE

#	Current State	Event /Condition =>Action	Next State
32	IDLE	**RPC_Release.ind ()** /CheckRPC () == InValid, ErrorDecode, ErrorCode1, ErrorCode2 => encode RPC Error PDU RPC_Release.rsp (-)	IDLE
33	IDLE	**RM_Release.rsp (+)** => FreeResource () encode RPC PDU RPC_Release.rsp (+)	IDLE
34	IDLE	**RM_Release.rsp (-)** => FreeResource () encode Error PDU RPC_Release.rsp (-)	IDLE
35	IDLE	**CMDEV_state_ind ()** /state == ABORT && CheckAREP () == Known => FreeResource ()	IDLE
36	IDLE	**CMDEV_state_ind ()** /state == ABORT && CheckAREP () == Unknown => ignore	IDLE
37	IDLE	**CMDEV_state_ind ()** /state != ABORT => ignore	IDLE
38	IDLE	**RPC_IOXSecure.ind ()** /CheckRPC () == Valid, UnknownAR && CheckResource () == Unavailable => ErrorCode2 := NO_AR_RESOURCE encode RPC Error PDU RPC_IOXSecure.rsp (-)	IDLE
39	IDLE	**RPC_IOXSecure.ind ()** /CheckRPC () == Valid, UnknownAR && CheckResource () == Available => RM_IOXSecure_ind ()	IDLE
40	IDLE	**RPC_IOXSecure.ind ()** /CheckRPC () == Valid, ExplicitAR => RM_IOXSecure_ind ()	IDLE
41	IDLE	**RPC_IOXSecure.ind ()** /CheckRPC () == Valid, ImplicitAR => // NOTE secure read implicit not possible ErrorCode2 := StateConflict encode RPC Error PDU RPC_IOXSecure.rsp (-)	IDLE
42	IDLE	**RPC_IOXSecure.ind ()** /CheckRPC () == InValid, ErrorDecode, ErrorCode1, ErrorCode2 => encode RPC Error PDU RPC_IOXSecure.rsp (-)	IDLE
43	IDLE	**RM_IOXSecure_rsp (+)** => encode RPC PDU RPC_IOXSecure.rsp (+)	IDLE

#	Current State	Event /Condition =>Action	Next State
44	IDLE	**RM_IOXSecure_rsp (-)** => encode RPC Error PDU RPC_IOXSecure.rsp (-)	IDLE

5.6.3.12.5 Functions, Macros, Timers and Variables

Table 1116 and Table 1117 contain the functions, macros, timers and variables used by the CMRPC and their arguments and their descriptions.

Table 1116 – Functions, Macros, Timers and Variables used by the CMRPC

Name	Type	Function/Meaning
AllocateResource	Function	This local function allocates the necessary resources for the AR using ARType, number of IOCRs and amount of IO data. An AR of the ARType IOCARSR allocates the resources for the complete ARset.
ARstate	Function	This local function defines the start value for the ARstate of the established AR. NONE: Variable not used for this type of AR. FIRST: Identifies the first established AR of an ARset BACKUP: Identifies the other established AR of an ARset
CheckAREP	Function	This local function checks whether the AREP exists.

Name	Type	Function/Meaning
CheckResource	Function	This local function checks the availability of the necessary resources for the connect indication using ARType, number of IOCRs and amount of IO data. Special case "Out of ARset resources" For the first application relation having ARType IOCARSR and belonging to a new ARSet (as indicated by the ARUUID) the resources of the complete ARSet are allocated; for example for all ARs of the complete ARSet. The resources of an application relation belonging to the same ARSet established at some later point are assigned from this allocated memory. The resources of the ARSet are freed when the last application relation of the ARSet is terminated. Special case "ARType specific resource handling" The resource checking shall cover the resource model shown in the GSD which optimizes the usage of the existing resources of an IO device. Special case “Dynamic reconfiguration” At the maximum two ARUUID.ConfigIDs for one ARset shall be active concurrently. One of the other ARs of the ARset shall be in ARstate := PRIMARY otherwise an IO device may reject an AR. Used ErrorCode2 := “ARset – State conflict during connection establishment” Special case “Port and interface resources” An IO device may reject an AR if the port or interface submodules are already owned by a different AR. Used ErrorCode2 := “Pdev already owned” Special case “Inconsistent interface and ports” An IO device may reject an AR if ports without the interface submodule are within the List of Expected Submodule. Used ErrorCode2 := “Pdev, port(s) without interface” Special case “Pdev ownership mismatch” An IO device may reject an AR with ARProperties (time-aware system) if Pdev ownership for the time-aware system is not given. Used ErrorCode2 := "Pdev, discrepancy between Pdev ownwership and ARProperties" Special case “R1 / R2 NAPs / DAPs” An IO device may reject an AR with ARType IOCAR if the addressed NAP contains the 0x8ipp submodules. Used ErrorCode2 := "Discrepancy between Pdev and ARType"

Name	Type	Function/Meaning
CheckRPC	Function	This local function checks the block structure and parameters against the definitions of this document. Special case "Shared IO device" The first established AR defines the SendClockFactor of the physical device. If a succeeding AR owns an unreachable SendClockFactor, this function returns InValid and a parameter error for the SendClockFactor or the ReductionRatio shall be created in the connect response negative. "Unreachable" may be a different SendClockFactor or a not representable combination of SendClockFactor and ReductionRatio. Special case "Shared IO device with RT_CLASS_3" The first established AR defines the IRDataUUID of the physical device. If a succeeding AR owns a different IRDataUUID, this function returns InValid and a parameter error for the IRDataUUID shall be created in the connect response negative. Special case "Startup of the ARs of an ARset" If the CMDEV already is establishing one AR of the set, then the CMRPC rejects a new AR connection request until ApplicationReady.cnf is reached. Also, an AR connection request of a new AR of the set is rejected if the last AR of the set is shutting down. Used ErrorCode2 := "ARset – State conflict during connection establishment" Special case "RPC detects a re-run" RPC checks whether a received service is a new request or a repetition due to, for example, network problems.
FreeResource	Function	This local function frees the allocated resources of the AR.
Encode Error PDU	Macro	This local macro generates the appropriate PDU with the detected errors from the PDU checking rules.
Encode RPC Error PDU	Macro	This local macro generates the appropriate PDU with the detected errors from the PDU checking rules. An RPC error PDU consists only of the RPC NDR header.
Encode RPC PDU	Macro	This local macro generates the appropriate PDU.
LocateAR	Macro	This local macro retrieves the associated AREP
Retrieve stored response	Macro	This local macro retrieves the previously stored parameter in case of a detected RPC rerun.
Store response	Macro	This local macro stores the parameters of the RPC response to be retrievable in case of a RPC rerun
AREP	Variable	This local variable stores the application relation endpoint for further use

Table 1117 – Return values of CheckRPC

Name	Function/Meaning
Valid	Valid states that the frame is correct according to this document.
InValid	InValid states that the frame is incorrect. Additionally, the ErrorDecode, ErrorCode1 and ErrorCode2 according to 5.2.6 are returned.
ExplicitAR	This AR, identified by the ARUUID, already exists.
UnknownAR	This AR, identified by the ARUUID, does not exist.
ImplicitAR	This service uses the implicit access to the device.
UnknownARset	This AR, identified by the ARUUID, is part of an ARset which does not exist.
ExplicitARset	This AR, identified by the ARUUID, is part of an ARset which already exists.
NoARset	This AR, identified by the ARUUID, is not part of an ARset.
Rerun	RPC detects a re-run due to, for example, network problems and reacts according to its specification.

5.6.3.12.6 Additional rules for Dynamic reconfiguration

Figure 212 shows the intersection and the residual amount of the ARs of an ARset.

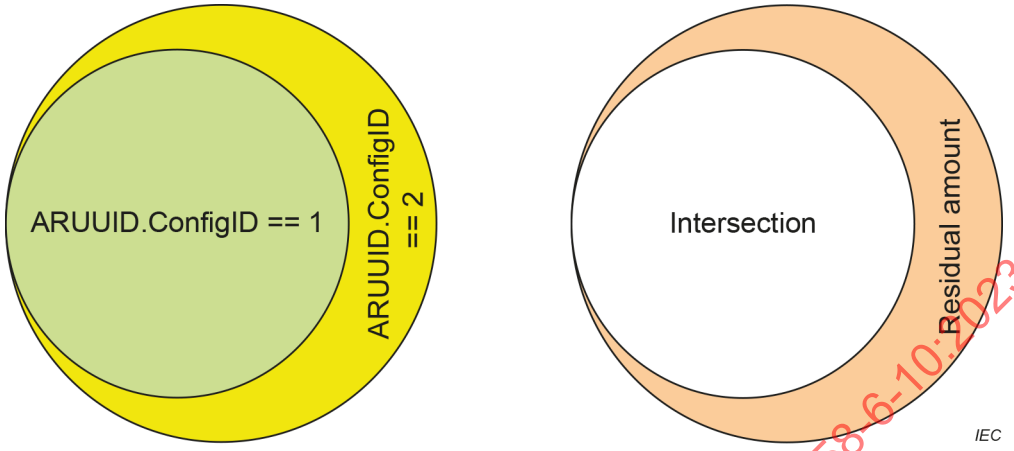


Figure 212 – Intersection and residual amount using different ARUUID.ConfigIDs

Intersection:

The standard rules apply.

Residual amount:

The IODConnect.rsp and the ControlBlockConnect(ApplRdy).req of the AR with ARUUID.ConfigID == 2 shall contain the residual amount of the Submodules stated as “NoSubmodule (NO)”.

If this AR is switched to ARstate == Primary the residual amount of Submodules shall be reported to the IO Controller using PlugAlarm.

Dynamic reconfiguration supports also the removal of submodules. This is shown in Figure 213. A combination of both scenarios (adding and removal of submodules) is also possible.

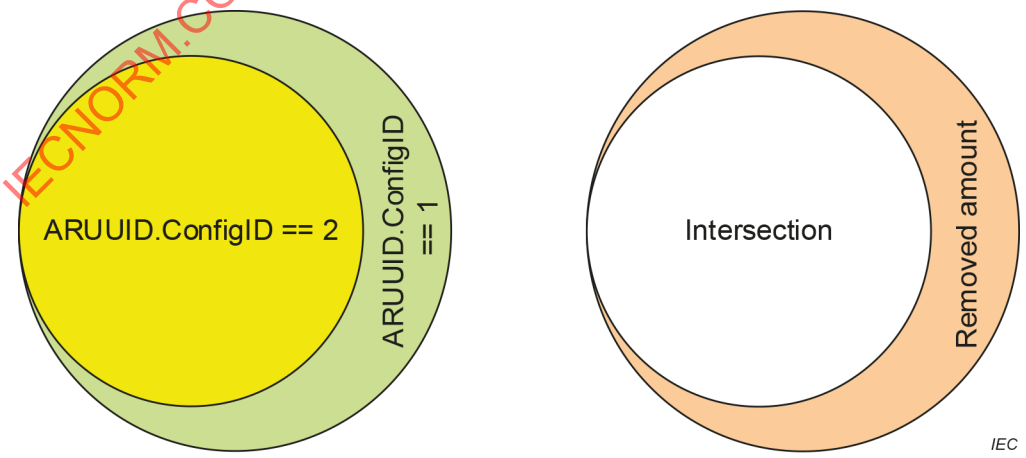


Figure 213 – Intersection and removed amount using different ARUUID.ConfigIDs

The standard rules for the intersection apply.

5.6.3.13 Context Management System Redundancy Layer Device

5.6.3.13.1 Primitive definitions

5.6.3.13.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management System Redundancy Layer Device Protocol Machine (CMSRL) are described in the service definition and shown in Table 1118.

Table 1118 – Remote primitives issued or received by CMSRL

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

5.6.3.13.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMSRL are described in the service definition and shown in Table 1119.

Table 1119 – Local primitives issued or received by CMSRL

Primitive	Source	Destination	Associated parameters	Functions
CMDEV_state_ind	CMDEV	CMSRL	state {ABORT, STARTUP, PRMEND, APPLRDY, DATA}	This service primitive controls the state of the AR establishment.
CMSRL_state_ind	CMSRL	CMSRL	AR#, state {PRIMARY, BACKUP}	This service primitive controls the state of an AR of an ARset.
CPM_NewData_ind	CPM	CMSRL	DataStatus {PrimaryRequest, BackupRequest}	This service primitive signals the DataStatus of a CPM.
PPM_Set_Status_req	CMSRL	PPM	DataStatus {PrimaryAcknowledge, PrimaryFault, BackupAcknowledge, PrimaryMissing}	This service primitive controls the DataStatus of a PPM.

5.6.3.13.2 State transition diagram

The state transition diagram of the CMSRL is shown in Figure 214.

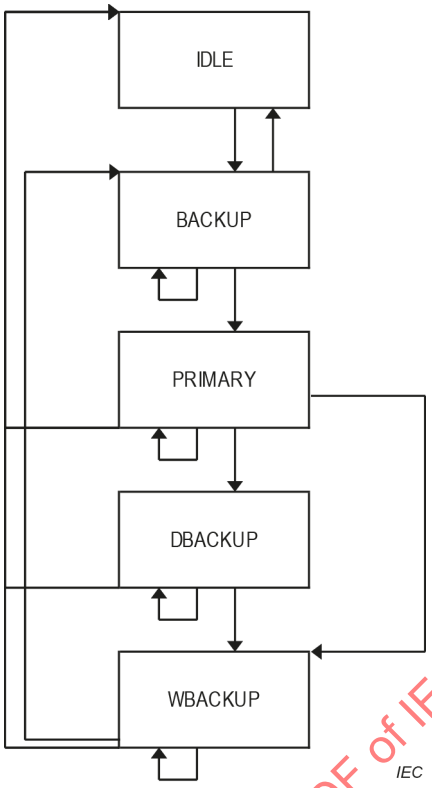


Figure 214 – State transition diagram of CMSRL

States of the CMSRL are:

IDLE	Waiting for an AR of an ARset.
PRIMARY	ARstate is Primary.
DBACKUP	ARstate from the IOD point of view is Backup. The IOC, supposing to be Primary, is informed about the difference between IOD and IOC and will clean up accordingly.
WBACKUP	ARstate is Backup. The IOD prepares the AR for the next switch to Primary before it acknowledges.
BACKUP	ARstate is Backup. The IOD is ready for a switchover.

5.6.3.13.3 State machine description

The CM System Redundancy Layer protocol machine (CMSRL) is present for each AR of the supported ARsets of an IO Device.

It handles the behavior of an IO Device in case of system redundancy and dynamic reconfiguration.

PrmBegin shall only be issued in state PRIMARY or DBACKUP.

Furthermore, the system redundancy data hold timer (RDHT) of an ARset is controlled by this state machine.

FAL user shall provide valid data (same or newer than the previous primary AR) before acknowledge switching BACKUP to PRIMARY.

5.6.3.13.4 CMSRL state table

Table 1120 contains the complete description of the CMSRL state machine.

Table 1120 – CMSRL state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**CMDEV_state_ind ()** /State == DATA && ARType == IOCARSR => ARstate := BACKUP	BACKUP
2	BACKUP	**CPM_NewData_ind ()** /EvaluateDataStatus () == BackupRequest => ignore	BACKUP
3	BACKUP	**CPM_NewData_ind ()** /EvaluateDataStatus () == PrimaryRequest => PrepareDataBuffers (PRIMARY) AssignAlarmChannels () StartTimer (RDHT) ARstate := PRIMARY UpdateDataTransferControl () CMSRL_State_ind (AR#, PRIMARY) PPM_Set_Status_req (PrimaryAcknowledge)	PRIMARY
4	BACKUP	**TimerExpired ()** => StopTimer (RDHT) CMDEV_state_ind (ABORT) //NOTE ARset aborted	IDLE
5	BACKUP	**ARsetStateChanged ()** /ARsetState == PrimaryExist => PPM_Set_Status_req (BackupAcknowledge)	BACKUP
6	BACKUP	**ARsetStateChanged ()** /ARsetState == PrimaryLost => PPM_Set_Status_req (PrimaryMissing)	BACKUP
7	PRIMARY	**CPM_NewData_ind ()** /EvaluateDataStatus () == PrimaryRequest => StartTimer (RDHT)	PRIMARY
8	PRIMARY	**TimerExpired ()** /ARset => StopTimer (RDHT) ARstate = BACKUP CMDEV_state_ind (ABORT) //NOTE ARset aborted	IDLE
9	PRIMARY	**CMDEV_state_ind ()** /State == ABORT && LastAROfARset == TRUE => StopTimer (RDHT)	IDLE
10	PRIMARY	**CMDEV_state_ind ()** /State == ABORT && LastAROfARset == FALSE => ignore	IDLE

#	Current State	Event /Condition =>Action	Next State
11	PRIMARY	**CMSRL_State_ind ()** /State == PRIMARY //NOTE Other AR is now Primary => CleanAlarmChannels () ARstate := BACKUP UpdateDataTransferControl () CMSRL_State_ind (AR#, BACKUP) PPM_Set_Status_req (PrimaryFault)	DBACKUP
12	PRIMARY	**CPM_NewData_ind ()** /EvaluateDataStatus () == BackupRequest => PrepareDataBuffers (BACKUP) CleanAlarmChannels () ARstate := BACKUP UpdateDataTransferControl () CMSRL_State_ind (AR#, BACKUP)	WBACKUP
13	PRIMARY	**ARsetStateChanged ()** => ignore	PRIMARY
14	DBACKUP	**CPM_NewData_ind ()** /EvaluateDataStatus () == BackupRequest => PrepareDataBuffers (BACKUP)	WBACKUP
15	DBACKUP	**TimerExpired ()** => StopTimer (RDHT) ARstate = BACKUP CMDEV_state_ind (ABORT) //NOTE ARset aborted	IDLE
16	DBACKUP	**CMDEV_state_ind ()** /State == ABORT && LastAROfARset == TRUE => StopTimer (RDHT) ARstate = BACKUP	IDLE
17	DBACKUP	**CMDEV_state_ind ()** /State == ABORT && LastARofARset == FALSE => ignore	IDLE
18	DBACKUP	**CPM_NewData_ind ()** /EvaluateDataStatus () == PrimaryRequest => ignore	DBACKUP
19	DBACKUP	**ARsetStateChanged ()** => ignore	DBACKUP
20	WBACKUP	**PreparationDone ()** /ARsetState == PrimaryExist => DetachAlarmChannels () PPM_Set_Status_req (BackupAcknowledge)	BACKUP
21	WBACKUP	**PreparationDone ()** /ARsetState == PrimaryLost => DetachAlarmChannels () PPM_Set_Status_req (PrimaryMissing)	BACKUP

#	Current State	Event /Condition =>Action	Next State
22	WBACKUP	**CMSRL_State_ind ()** /State == PRIMARY //NOTE Other AR is now Primary => ignore	WBACKUP
23	WBACKUP	**CPM_NewData_ind ()** /EvaluateDataStatus () == PrimaryRequest && LastAROfARset == TRUE => StopTimer (RDHT) ARstate := BACKUP CMDEV_state_ind (ABORT) //NOTE ARset aborted	IDLE
24	WBACKUP	**CPM_NewData_ind ()** /EvaluateDataStatus () == PrimaryRequest && LastAROfARset == FALSE => ARstate := BACKUP CMDEV_state_ind (ABORT) //NOTE AR aborted	IDLE
25	WBACKUP	**ARsetStateChanged ()** => ignore	WBACKUP

5.6.3.13.5 Functions, Macros, Timers and Variables

5.6.3.13.5.1 General

Table 1121, Table 1122, and Table 1123 contain the functions, macros, timers and variables used by the CMSRL and their arguments and their descriptions.

Table 1121 – Functions, Macros, Timers and Variables used by the CMSRL

Name	Type	Function/Meaning
ARsetStateChanged	Function	This ARset global function is issued if the state of the ARset changes. See variable "ARsetState"
AssignAlarmChannels	Function	This local function handles the selection of the Alarm channels for the FSPM if an ARstate changes to Primary. See 5.6.3.13.7.
CleanAlarmChannels	Function	This local function handles the cleanup of the Alarm channels for the FSPM if an ARstate changes to Backup. See 5.6.3.13.7.
DetachAlarmChannels	Function	This local function handles the deselection of the Alarm channels for the FSPM if an ARstate changes to Backup. See 5.6.3.13.7.
EvaluateDataStatus	Function	This local function is issued to check the DataStatus from the CPM.

Name	Type	Function/Meaning
PreparationDone	Function	This local function is issued if the preparation for the BackupAcknowledge/PrimaryMissing is done. The preparation is done, when the Alarm channels are cleaned (no pending AlarmAck) and detachable, and if needed, the Input buffer is updated to allow a fast switchover to Primary.
PrepareDataBuffers	Function	This local function handles the management of the Input and Output data buffers of the AR of an ARset. See 5.6.3.13.6. The Input buffer signaling PrimaryAcknowledge shall always contain the newest available data.
StartTimer	Function	This local function starts or restarts a timer.
StopTimer	Function	This local function stops a timer.
TimerExpired	Function	This local function is issued if a timer expires.
UpdateDataTransferControl	Function	This ARset global function is issued to update the DataTransferControl for all CPMs of the ARs of the ARset. ARstate==Backup DataTransferControl for all CPMs are set to Discard. ARstate==Primary DataTransferControl for the CPM is set to Transfer. Transition from Backup->Primary First the CPM that is no longer used as Primary is set to Discard, then the CPM that is now Primary is set to Transfer.
RDHT	Timer	This local timer is used to monitor an ARset. It is started or restarted with the RDHT value of the AR in ARstate==Primary If RDHT expires, abort reason “IOCARSR – RDHT expired” shall be used.
ARsetState	Variable	This ARSet global variable is used to store the ARsetState information. PrimaryLost: No AR of the ARset is in ARstate==Primary PrimaryExist: One AR of the ARset is in ARstate==Primary This variable starts with value IDLE. Thus, PrimaryMissing shall only be signaled after the first transition from PrimaryExist to PrimaryLost.
ARstate	Variable	This local variable is used to store the ARstate information.
LastAROfARset	Variable	This local variable is used to store the information whether this AR is the last existing AR of the ARset. TRUE: Only one AR of an ARset exists. Thus the ARset resources shall be cleaned up after this AR is released. FALSE: More than one AR of an ARset exist. Thus the ARset resources are needed even if this AR is released.

Table 1122 – Combinations of DataStatus for Output buffers

APDU_Status.DataStatus			Function/Meaning
State	Redundancy	DataValid	
Primary	Zero	Valid	PrimaryRequest
Backup	Zero	Valid	BackupRequest

Table 1123 – Combinations of DataStatus for Input buffers

APDU_Status.DataStatus			Function/Meaning
State	Redundancy	DataValid	
Backup	Primary present	Valid	BackupAcknowledge with actual data independent of the ARstate
Backup	Primary present	Invalid	BackupAcknowledge without actual data
Backup	Primary missing	Valid	PrimaryMissing with actual data independent of the ARstate NOTE Read as BackupAcknowledge with PrimaryMissing
Backup	Primary missing	Invalid	PrimaryMissing without actual data NOTE Read as BackupAcknowledge with PrimaryMissing
Primary	IO device view is primary	Valid	PrimaryAcknowledge
Primary	IO device view is backup	Valid	PrimaryFault

5.6.3.13.5.2 Special case multiple CRs per AR

This document offers the possibility to use multiple CRs per AR. If this feature is combined with CMSRL, the following rules applies:

A valid PrimaryRequest is detected if all OUTPUT CRs of an AR contain it.

– State Primary is only achieved if all outputs are valid!
– The switchover cannot be reversed. If one CR starts, all other CRs of the AR shall follow

A valid BackupRequest is detected if one OUTPUT CRs of an AR contains it.

– State Primary is lost (PrimaryMissing), when one output is no longer valid (BackupRequest)
– The switchover cannot be reversed. If one CR starts, all other CRs of the AR shall follow
– BackupAcknowledge shall only be signaled if all CRs of the AR contain BackupRequest

The RDHT is only triggered if all outputs are valid.

– The RDHT is not triggered during PrimaryMissing.

5.6.3.13.6 Buffer management of CMSRL

5.6.3.13.6.1 Resource model

The principle of the Input and Output buffer management for CMSRL is shown in Figure 215.

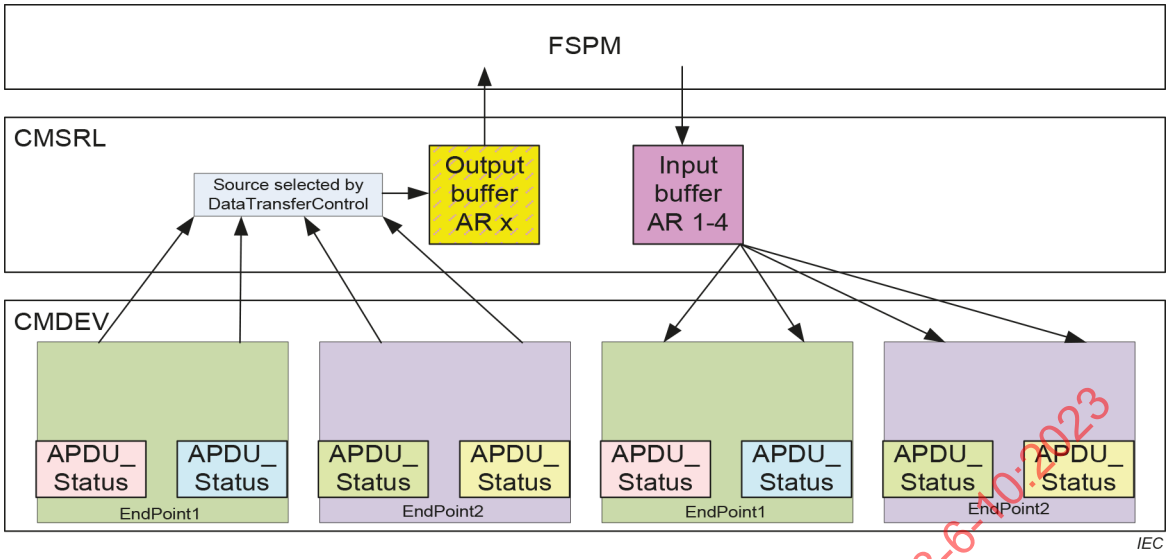


Figure 215 – Single Input and single Output buffer of CMSRL

The Input and Output buffers conveyed in each AR of the ARset contain the identical data structure and length. Thus, a single Input and Output buffer with multiple APDU_Status is maintained.

The CPM buffer update rules, extended by DataTransferControl, valid for the Output buffer from the IO device point of view, are given in Table 244.

Special case: More than one input buffer

The FSPM shall update the data of the Input buffer of the AR in ARstate == Primary or update all Input buffers if SRProperties.InputValidOnBackupAR == TRUE.

An update of the Input buffers for the ARs in ARstate == Backup if SRProperties.InputValidOnBackupAR == FALSE is superfluous due to APDU_Status.DataStatus.DataValid = Invalid.

5.6.3.13.6.2 Constraints for Dynamic reconfiguration

For dynamic reconfiguration, during the transition from one ARUUID.ConfigID to another, as shown in Figure 216, at least two Input buffers and two Output buffers are needed, one for the ARs with the previous ARUUID.ConfigID and one for the ARs with the current ARUUID.ConfigID.

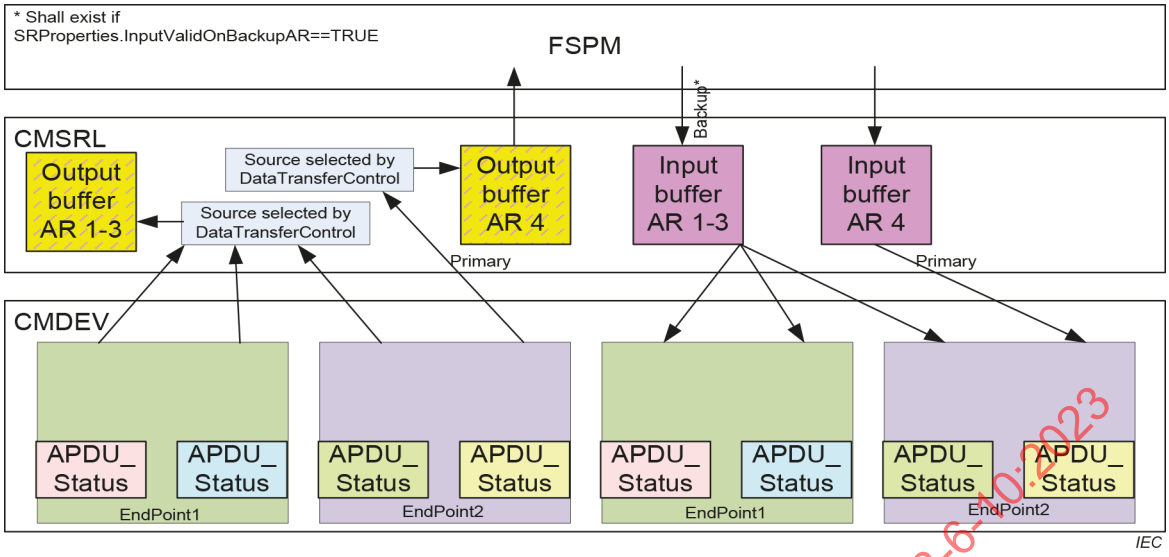


Figure 216 – Dynamic reconfiguration with CMSRL

The FSPM shall update the data of the Input buffer of the AR in ARstate == Primary or update all Input buffers if SRProperties.InputValidOnBackupAR == TRUE.

An update of the Input buffers for the ARs in ARstate == Backup if SRProperties.InputValidOnBackupAR == FALSE is superfluous due to APDU_Status.DataStatus.DataValid = Invalid.

5.6.3.13.6.3 FSPM access rules

The FSPM

– initializes the Output buffer of the ARset at startup, and
– updates the Input buffer independently from the ARstate.

No additional access rules are needed for an ARset.

5.6.3.13.7 Alarm queue management of CMSRL

The principle of the Alarm queue management for CMSRL is shown in Figure 217.

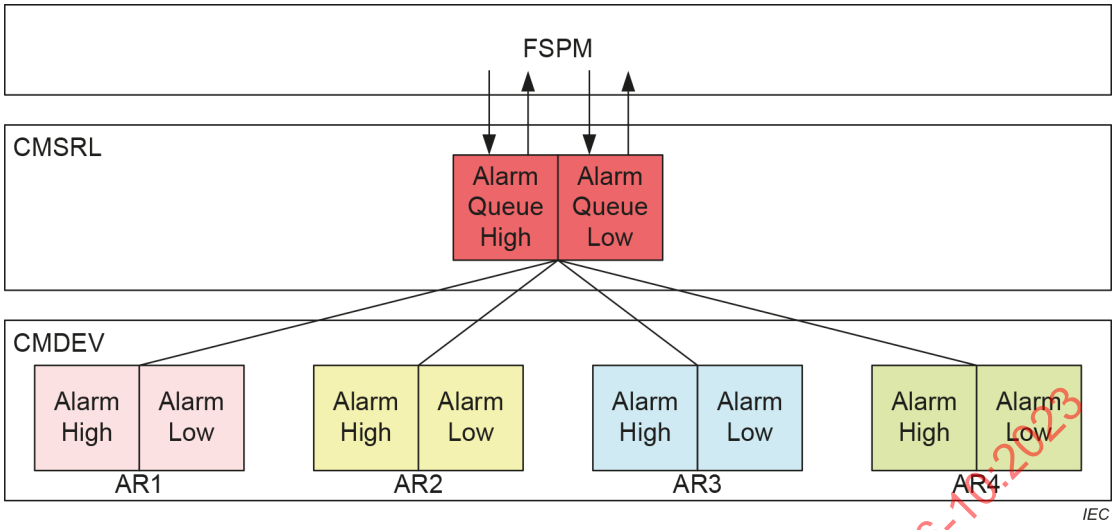


Figure 217 – Alarm queue management of CMSRL

The CMSRL maintains a queue of the depth one for each Alarm priority and attaches it to the AR in ARstate == Primary.

If no AR in ARstate == Primary exists, then the CMSRL waits for an AR in ARstate == Primary to attach it to.

As Alarm source (see 5.6.3.14), the CMSRL stores one outstanding request for each Alarm priority for retransmission (using the same AlarmSpecifier.SequenceNumber) until an AR enters ARstate == Primary and this Alarm is acknowledged in ARstate == Primary.

An acknowledge with “IOCARSR: Backup – Alarm not executed” in ARstate == Primary is the trigger for storing the request as an “outstanding request waiting for retransmission”. The alarm queue is now blocked until the stored request is retransmitted. The stored request shall be retransmitted as soon as the queue is attached to the new AR in ARstate == Primary.

As Alarm sink, CMSRL in ARstate == Backup acknowledges outstanding requests with “IOCARSR: Backup – Alarm not executed” to clean the previous Primary AR.

In ARstate == Backup, all alarms received from CMDEV are rejected using “IOCARSR: Backup – Alarm not executed”, and all alarms received from the FSPM are rejected by local means.

Special case:
Changes of the RealIdentification due to plug or pull of a real Submodule can lead to multiple IO device internal events, due to mechanical reasons. As a result, different values for the AlarmSequenceNumber can be seen for one plug or pull of a real Submodule (same for “Controlled by supervisor”, “Released”, and “Plug Wrong Submodule”).

Plug or pull activated additional state machines are described in IEC 61158-5-10. This activation is done as soon as the associated ALARM is scheduled for transmission. In case of retransmission using the new AR in ARstate == Primary, the associated state machine is started again.

5.6.3.13.8 Reporting System management of CMSRL

5.6.3.13.8.1 Resource model

The principle of the Reporting System management for CMSRL is shown in Figure 217.

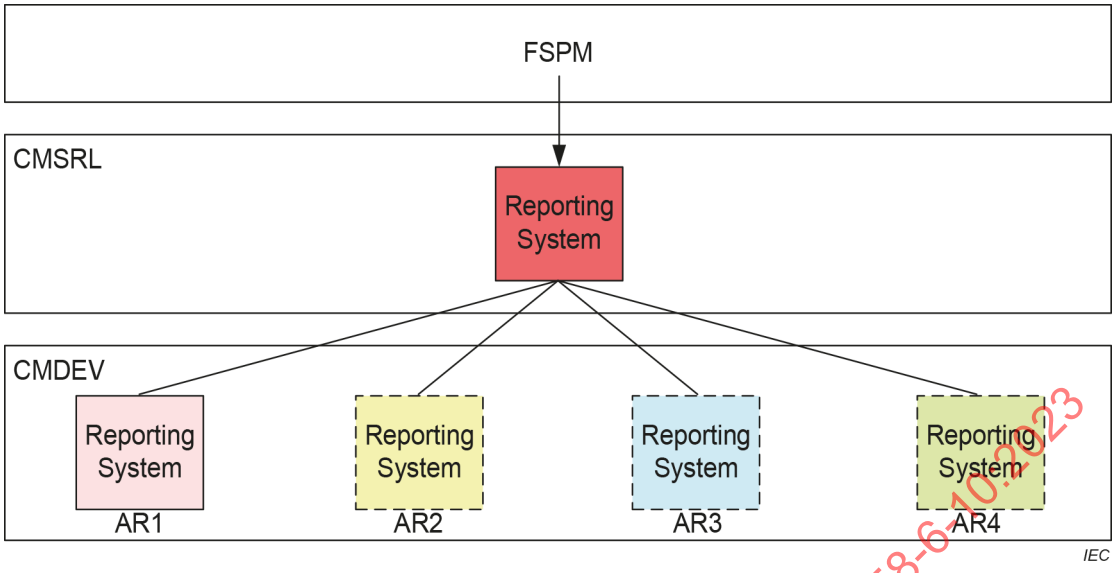


Figure 218 – Reporting System management of CMSRL

The CMSRL attaches the Reporting System to the AR in ARstate == Primary.

If no AR in ARstate == Primary exists, then the CMSRL attaches it to the first connected AR. In the following text, ARstate == Primary is used for both, Primary and first connected if no Primary exists.

5.6.3.13.8.2 Constraints for Dynamic reconfiguration

For dynamic reconfiguration, during the transition from one ARUUID.ConfigID to another, the Reporting System may be added or removed.

5.6.3.13.9 Switchover time between two ARs of an ARset

The switchover time between two ARs of an ARset is measured according to Figure 219 and Figure 220.

The calculation of Maximum Switchover Time MSOT is done according to Formula (86) and Formula (87).

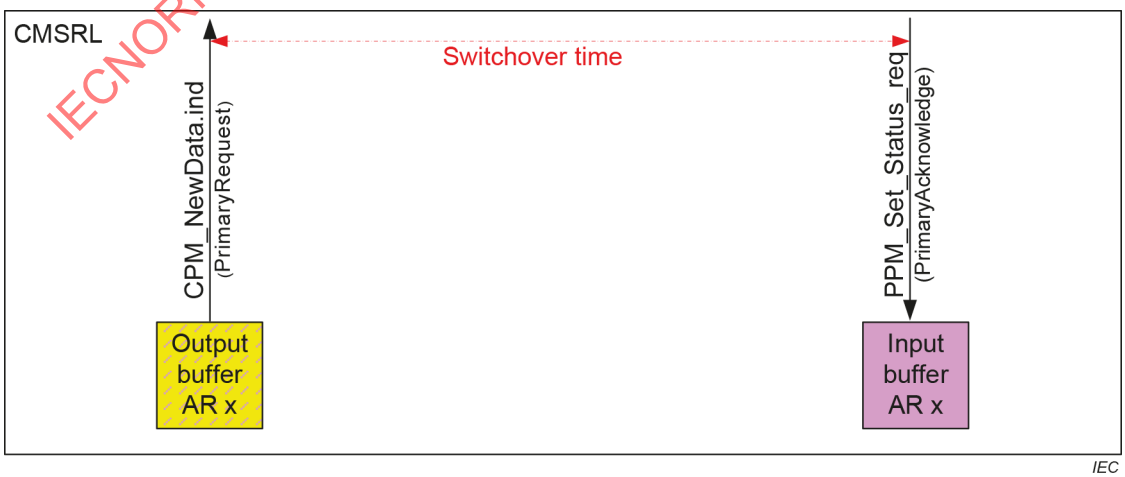


Figure 219 – Primary: Switchover time between two ARs of an ARset

$$MSOT\_primary = maximum (TimeStamp(PPM\_Set\_Status\_req (PrimaryAcknowledge)) – TimeStamp(CPM\_NewData.ind(PrimaryRequest))) \quad (86)$$

where

MSOT_primary	is the Maximum Switchover Time
maximum	is a function returning the maximum value for a given implementation
TimeStamp	is a function returning the point in time when the event occurs
CPM_NewData.ind (PrimaryRequest)	is the event defining the start of switchover
PPM_Set_Status_req (Primary*Acknowledge*)	is the event defining the end of switchover

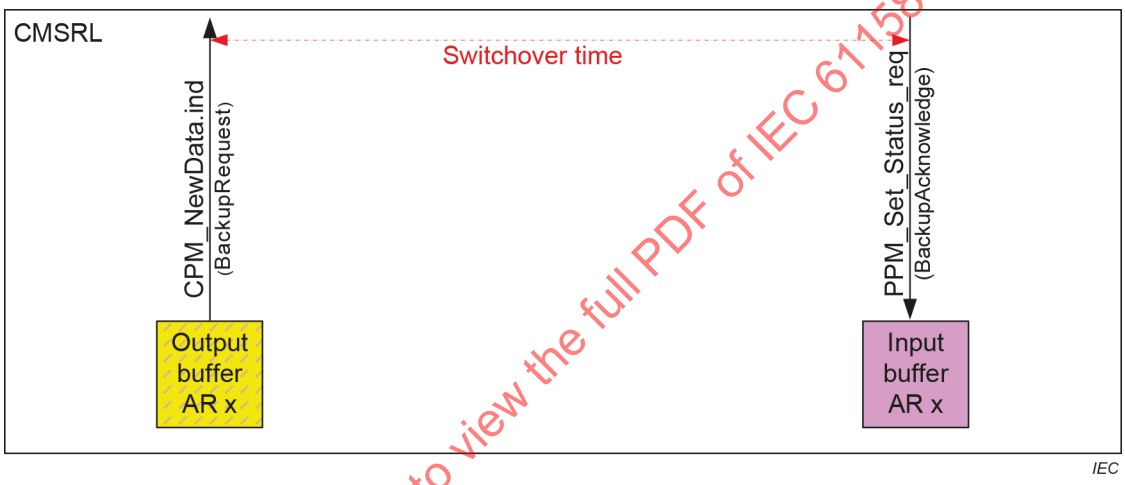


Figure 220 – Backup: Switchover time between two ARs of an ARset

$$MSOT\_backup = maximum (TimeStamp(PPM\_Set\_Status\_req (BackupAcknowledge)) – TimeStamp(CPM\_NewData.ind(BackupRequest))) \quad (87)$$

where

MSOT_backup	is the Maximum Switchover Time
maximum	is a function returning the maximum value for a given implementation
TimeStamp	is a function returning the point in time when the event occurs
CPM_NewData.ind (BackupRequest)	is the event defining the start of switchover
PPM_Set_Status_req (BackupAcknowledge)	is the event defining the end of switchover

5.6.3.14 Context Management System Redundancy Layer Alarm Device

5.6.3.14.1 Primitive definitions

5.6.3.14.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management System Redundancy Layer Alarm Device Protocol Machine (CMSRL_AL) are described in the service definition and shown in Table 1124.

Table 1124 – Remote primitives issued or received by CMSRL_AL

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

5.6.3.14.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMSRL_AL are described in the service definition and shown in Table 1125.

Table 1125 – Local primitives issued or received by CMSRL_AL

Primitive	Source	Destination	Associated parameters	Functions
ForwardAlarm_req	CMSRL_AL	—	—	The CMSRL_AL layer forwards an Alarm.req.
IssueAlarm_cnf (-)	CMSRL_AL	—	—	The CMSRL_AL layer rejects an Alarm.req according to the SRL definitions.
ForwardAlarm_cnf	—	CMSRL_AL	PNIOStatus	The CMSRL_AL layer receives an Alarm.cnf.
IssueAlarm_req	—	CMSRL_AL	—	The CMSRL_AL layer receives an Alarm.req and handles it according to the SRL definitions.

5.6.3.14.2 State transition diagram

The state transition diagram of the CMSRL_AL is shown in Figure 221.

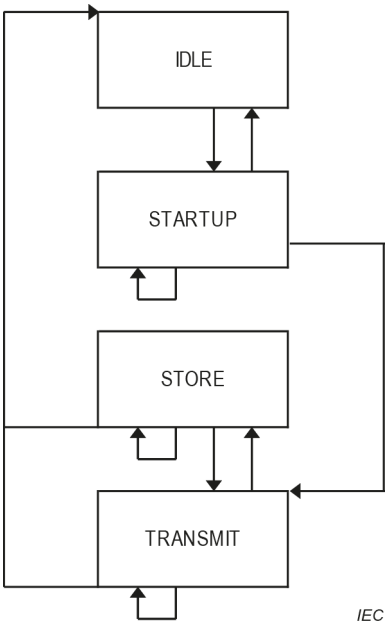


Figure 221 – State transition diagram of CMSRL_AL

States of the CMSRL_AL are:

IDLE	Waiting for an ARset.
STARTUP	An ARset is established. Transmission of Alarms before ARstate:=Primary is disabled.
STORE	The ARset is in an intermediate state during switchover. Transmission or retransmission of Alarms is not allowed.
TRANSMIT	Transmission or retransmission of Alarms is allowed.

5.6.3.14.3 State machine description

The CM System Redundancy Layer Alarm Device protocol machine (CMSRL_AL), subsequent to the CMSRL, is present for each ARset of an IO Device.

It handles the behavior of the Alarm transmission in case of system redundancy and dynamic reconfiguration.

An instance exists per priority (one for ALARM high and one for ALARM low).

5.6.3.14.4 CMSRL_AL state table

Table 1126 contains the complete description of the CMSRL_AL state machine.

Table 1126 – CMSRL_AL state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	=> ignore	STARTUP
2	STARTUP	**ARsetTerminated ()** => ignore	IDLE
3	STARTUP	**ARsetStateChanged ()** /ARsetState == PrimaryExist => ignore	TRANSMIT
4	STARTUP	**IssueAlarm_req ()** => IssueAlarm_cnf (-) //NOTE Alarm rejected	STARTUP
5	STARTUP	**ForwardAlarm_cnf ()** => ignore	STARTUP
6	STORE	**ARsetTerminated ()** => ignore	IDLE
7	STORE	**ARsetStateChanged ()** /ARsetState == PrimaryExist && AlarmStored == TRUE => StoreAlarm () ForwardAlarm_req ()	TRANSMIT
8	STORE	**ARsetStateChanged ()** /ARsetState == PrimaryExist && AlarmStored == FALSE => ignore	TRANSMIT
9	STORE	**IssueAlarm_req ()** => StoreAlarm ()	STORE
10	STORE	**ForwardAlarm_cnf ()** => ignore	STORE
11	TRANSMIT	**ARsetTerminated ()** => ignore	IDLE
12	TRANSMIT	**ARsetStateChanged ()** /ARsetState == PrimaryLost => ignore	STORE
13	TRANSMIT	**IssueAlarm_req ()** => StoreAlarm () ForwardAlarm_req ()	TRANSMIT
14	TRANSMIT	**ForwardAlarm_cnf (PNIOStatus)** /PNIOStatus == "IOCARSR: Backup – Alarm not executed" => ignore	STORE
15	TRANSMIT	**ForwardAlarm_cnf (PNIOStatus)** /PNIOStatus != "IOCARSR: Backup – Alarm not executed" => IssueAlarm_cnf (+)	TRANSMIT

5.6.3.14.5 Functions, Macros, Timers and Variables

Table 1127 contains the functions, macros, timers and variables used by the CMSRL_AL and their arguments and their descriptions.

Table 1127 – Functions, Macros, Timers and Variables used by the CMSRL_AL

Name	Type	Function/Meaning
ARsetStateChanged	Function	This ARset global function is issued if the state of the ARset changes. See variable "ARsetState"
ARsetTerminated	Function	This ARset global function is issued if the state of the ARset changes. See variable "LastAROfARset"
StoreAlarm	Function	This local function is used to store one ALARM to be transmitted. See variable "AlarmStored"
AlarmStored	Variable	This local variable is used to store the information whether an ALARM is stored or not. TRUE: Stored ALARM exists. FALSE: No ALARM stored.
ARsetState	Variable	This ARSet global variable is used to store the ARsetState information. PrimaryLost: No AR of the ARset is in ARstate==Primary PrimaryExist: One AR of the ARset is in ARstate==Primary This variable starts with value IDLE. Thus, PrimaryMissing shall only be signaled after the first transition from PrimaryExist to PrimaryLost.
LastAROfARset	Variable	This local variable is used to store the information whether this AR is the last existing AR of the ARset. TRUE: Only one AR of an ARset exists. Thus the ARset resources shall be cleaned up after this AR is released. FALSE: More than one AR of an ARset exist. Thus the ARset resources are needed even if this AR is released.

5.6.3.15 Context Management RSI Device

5.6.3.15.1 Primitive definitions

5.6.3.15.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by Context Management RSI Device Protocol Machine (CMRSI) are described in the service definition and shown in Table 1128.

Table 1128 – Remote primitives issued or received by CMRSI

Primitive	Source	Destination	Associated parameters	Functions
RSI_Call.ind	RSIR	CMRSI	RSAP, OpNum, ArgsRspMax, ArgsReqLength, ArgsReq	—
RSI_Call.rsp	CMRSI	RSIR	RSAP, PNIOStatus, ArgsRspLength, ArgsRsp	—
RSI_I_Abort.ind	RSIR	CMRSI	RSAP PNIOStatus	—
RM_Connect.ind	CMRSI	CMDEV CMPBE	AREP, ARBlockReq, ListOfIOCRBlockReq, AlarmCRBlockReq, ListOfExpectedSubmoduleBlockReq	—
RM_Connect.rsp (-)	CMDEV	CMRSI	AREP, PNIOStatus	—
RM_Connect.rsp (+)	CMDEV	CMRSI	AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock	—
RM_Release.ind	CMRSI	CMDEV CMDEV_DA	AREP, ControlBlock	—
RM_Release.rsp	CMDEV CMDEV_DA	CMRSI	AREP	—
RM_Read.ind	CMRSI	CMSM CMRDR	AREP, API, TargetUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length	—
RM_Read.rsp (-)	CMRDR	CMRSI	AREP, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Read.rsp (+)	CMRDR	CMRSI	AREP, SeqNumber, AddData1, AddData2, Length, Data	—
RM_Write.ind	CMRSI	CMWRR	AREP, API, SlotNumber, SubslotNumber, Index, Multiple, SeqNumber, PrmEnd, Length, Data	—

Primitive	Source	Destination	Associated parameters	Functions
RM_Write.rsp (-)	CMWRR	CMRSI	AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Write.rsp (+)	CMWRR	CMRSI	AREP, SeqNumber, AddData1, AddData2	—
RM_Dcontrol.ind	CMRSI	CMDEV	AREP, ControlBlock	—
RM_Dcontrol.rsp (-)	CMDEV	CMRSI	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Dcontrol.rsp (+)	CMDEV	CMRSI	AREP, ControlBlock	—
RM_Ccontrol.req	CMDEV CMPBE	CMRSI	AREP, ControlBlock, ModuleDiffBlock	—
RM_Ccontrol.cnf (-)	CMRSI	CMDEV CMPBE	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Ccontrol.cnf (+)	CMRSI	CMDEV CMPBE	AREP, ControlBlock	—

5.6.3.15.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMRSI are described in the service definition and shown in Table 1129.

Table 1129 – Local primitives issued or received by CMRSI

Primitive	Source	Destination	Associated parameters	Functions
CMDEV_state_ind	CMRSI	CMDEV	AREP, state {ABORT }	—
RSI_Notification.req	CMRSI	RSIRN	RSAP, Mode	—
RSI_Notification.cnf	RSIRN	CMRSI	RSAP	—
RM_IOXSecure.ind	CMRSI	CMSAM	AREP, Arg	—
RM_IOXSecure.rsp	CMSAM	CMRSI	AREP, PNIOStatus, Arg	—

5.6.3.15.2 State transition diagram

The state transition diagram of the CMRSI is shown in Figure 222.

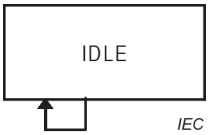


Figure 222 – State transition diagram of CMRSI

State of the CMRSI is:

IDLE Wait for an RM or an RSI call, check the integrity and the parameter.

5.6.3.15.3 State machine description

The Context Management RSI Device Protocol Machine (CMRSI) arranges the RSIR communication and handles all issued and received RSIR services.

5.6.3.15.4 CMRSI state table

Table 1130 contains the complete description of the CMRSI state machine.

Table 1130 – CMRSI state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**RSI_Call.ind** /OpNum == Connect && CheckRPC () == Valid, UnknownAR, UnknownARset => AllocateResource (RSAP) ARstate (FIRST) RM_Connect.ind	IDLE
2	IDLE	**RSI_Call.ind** /OpNum == Connect && CheckRPC () == Valid, UnknownAR, ExplicitARset //NOTE Preallocated resources by the first AR of an ARset => ARstate (BACKUP) RM_Connect.ind	IDLE
3	IDLE	**RSI_Call.ind** /OpNum == Connect && CheckRPC () == Valid, UnknownAR, NoARset => AllocateResource (RSAP) ARstate (NONE) RM_Connect.ind	IDLE
4	IDLE	**RSI_Call.ind** /OpNum == Connect && CheckRPC () == Valid, ExplicitAR => PNIOStatus := StateConflict encode RSI-RSP-PDU RSI_Call.rsp (-) CMDEV_state_ind (ABORT)	IDLE
5	IDLE	**RSI_Call.ind** /OpNum == Connect && CheckRPC () == InValid, ErrorDecode, ErrorCode1, ErrorCode2 => PNIOStatus := (Connect, ErrorDecode, ErrorCode1, ErrorCode2) RSI_Call.rsp (-)	IDLE

#	Current State	Event /Condition =>Action	Next State
6	IDLE	**RM_Connect.rsp (+)** => PNIOStatus := OK encode RSI-RSP-PDU RSI_Call.rsp (+)	IDLE
7	IDLE	**RM_Connect.rsp (-)** => PNIOStatus := PNIOStatus encode RSI-RSP-PDU RSI_Call.rsp (-)	IDLE
8	IDLE	**RSI_Call.ind** /OpNum == Read => AREP := LocateAR (RSAP) RM_Read.ind	IDLE
9	IDLE	**RM_Read.rsp (+)** => PNIOStatus := OK encode RSI-RSP-PDU RSI_Call.rsp (+)	IDLE
10	IDLE	**RM_Read.rsp (-)** => PNIOStatus := PNIOStatus encode RSI-RSP-PDU RSI_Call.rsp (-)	IDLE
11	IDLE	**RSI_Call.ind** /OpNum == Write => AREP := LocateAR (RSAP) PrmEnd := False RM_Write.ind	IDLE
12	IDLE	**RSI_Call.ind** /OpNum == PrmWriteMore => AREP := LocateAR (RSAP) PrmEnd := False RM_Write.ind	IDLE
13	IDLE	**RSI_Call.ind** /OpNum == PrmWriteEnd => AREP := LocateAR (RSAP) PrmEnd := True RM_Write.ind	IDLE
14	IDLE	**RM_Write.rsp (+)** /PrmEnd == False => PNIOStatus := OK encode RSI-RSP-PDU RSI_Call.rsp (+)	IDLE
15	IDLE	**RM_Write.rsp (+)** /PrmEnd == True => PNIOStatus := OK encode RSI-RSP-PDU RSI_Call.rsp (+) RM_Dcontrol.ind	IDLE
16	IDLE	**RM_Write.rsp (-)** /PrmEnd == False => PNIOStatus := PNIOStatus encode RSI-RSP-PDU RSI_Call.rsp (-)	IDLE

#	Current State	Event /Condition =>Action	Next State
17	IDLE	**RM_Write.rsp (-)** /PrmEnd == True => PNIOStatus := PNIOStatus encode RSI-RSP-PDU RSI_Call.rsp (-) RM_Dcontrol.ind	IDLE
18	IDLE	**RSI_Call.ind** /OpNum == Control => AREP := LocateAR (RSAP) RM_Dcontrol.ind	IDLE
19	IDLE	**RM_Dcontrol.rsp (+)** => PNIOStatus := OK encode RSI-RSP-PDU RSI_Call.rsp (+)	IDLE
20	IDLE	**RM_Dcontrol.rsp (-)** => PNIOStatus := PNIOStatus encode RSI-RSP-PDU RSI_Call.rsp (-)	IDLE
21	IDLE	**RM_Ccontrol.req** //* ApplicationReady */ => RSAP := LocateRSAP (AREP) RSI_Notification.req (Set)	IDLE
22	IDLE	**RSI_Call.ind** /OpNum == ReadNotification => AREP := LocateAR (RSAP) index := 0xE041 // ReadNotification RM_Ccontrol.cnf RSI_Notification.req (Reset) RM_Read.ind	IDLE
23	IDLE	**RM_Read.rsp (+)** /index == 0xE041 // ReadNotification => PNIOStatus := OK encode RSI-RSP-PDU RSI_Call.rsp (+)	IDLE
24	IDLE	**RM_Read.rsp (-)** /index == 0xE041 // ReadNotification => PNIOStatus := PNIOStatus encode RSI-RSP-PDU RSI_Call.rsp (-)	IDLE
25	IDLE	**RSI_Notification.cnf ()** => ignore	IDLE
26	IDLE	**RSI_I_Abort.ind ()** => AREP := LocateAR (RSAP) CMDEV_state_ind (ABORT)	IDLE
27	IDLE	**RSI_Call.ind** /OpNum == SecurityAssociationControl && CheckRPC () == Valid, UnknownAR => RM_IOXSecure.ind	IDLE
28	IDLE	**RSI_Call.ind** /OpNum == SecurityAssociationControl && CheckRPC () == Valid, ExplicitAR => RM_IOXSecure.ind	IDLE

#	Current State	Event /Condition =>Action	Next State
29	IDLE	**RSI_Call.ind** /OpNum == SecurityAssociationControl && CheckRPC () == Valid, ImplizitAR => // NOTE secure read implicit not possible PNIOStatus.ErrorCode2 := StateConflict RSI_Call.rsp (-)	IDLE
30	IDLE	**RSI_Call.ind** /OpNum == SecurityAssociationControl && CheckRPC () == InValid, ErrorDecode, ErrorCode1, ErrorCode2 => PNIOStatus := IOXSecure, ErrorDecode, ErrorCode1, ErrorCode2 RSI_Call.rsp (-)	IDLE
31	IDLE	**RM_IOXSecure.rsp (+)** => PNIOStatus := OK encode RSI-RSP-PDU RSI_Call.rsp (+)	IDLE
32	IDLE	**RM_IOXSecure.rsp (-)** => PNIOStatus := PNIOStatus encode RSI-RSP-PDU RSI_Call.rsp (-)	IDLE

5.6.3.15.5 Functions, Macros, Timers and Variables

Table 1131 contains the functions, macros, timers and variables used by the CMRSI and their arguments and their descriptions.

Table 1131 – Functions, Macros, Timers and Variables used by the CMRSI

Name	Type	Function/Meaning
AREP	Variable	This local variable stores the application relation endpoint for further use.
RSAP	Variable	This local variable stores the responder service access point for further use.
CheckRPC	Function	See CMRPC
AllocateResource	Function	See CMRPC
ARstate	Function	See CMRPC
LocateAR	Macro	This local macro retrieves the associated AREP.
LocateRSAP	Macro	This local macro retrieves the associated RSAP.
Encode RSI-RSP-PDU	Macro	This local macro generates the corresponding PDU.

5.6.4 Controller

5.6.4.1 General

5.6.4.1.1 General

The CMCTL arranges the connection establishment of an IO controller. Figure 223 shows the integration of the IO controller CM.

The extensions CIM and “Connection to CIM” shall be used for ARBlockReq.ARProperties.-TimeAwareSystem == TimeAware.

The extensions CIM and “Connection to CIM” shall be used for ARBlockReq.ARProperties.-ProtectionProperties != 0.

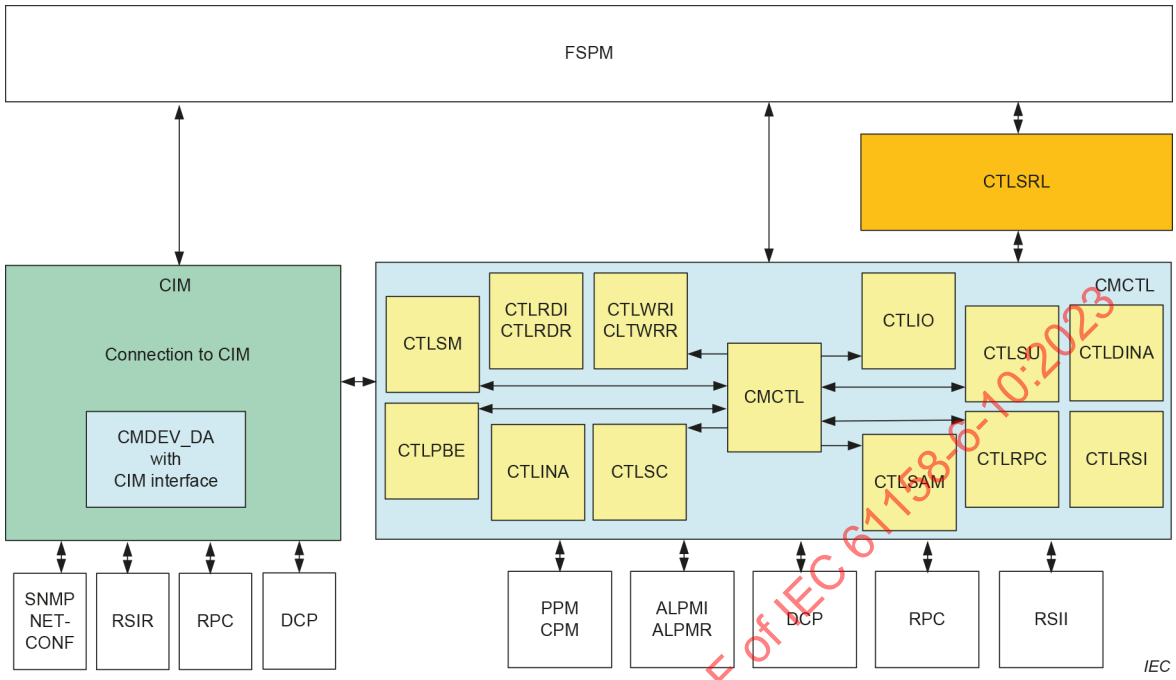


Figure 223 – Scheme of the IO controller CM

CMCTL
This state machine handles the context management of the IO controller. It contains the Query Stream functionality.

CTLWRI
This state machine handles the initiator functionality of the write service.

CTLWRR
This state machine handles the responder functionality of the write service.

CTLRDI
This state machine handles the initiator functionality of the read service.

CTLRDR
This state machine handles the responder functionality of the read service.

CTLSM
This state machine handles the connection monitoring during the startup.

CTLPBE
This state machine handles the PrmBegin, PrmEnd and ApplRdy sequence. It is the basis for system redundancy and dynamic reconfiguration.

CTLSU
This state machine handles the startup of the different state machines during startup and shutdown.

CTLRPC
This state machine handles the translation of CMCTL services to RPC services and vice versa.

CTLIO
This state machine handles the IO Data services.

CTLDINA
This machine handles the discovery, IP and station name assignment.

CTLSRL
This state machine handles higher reliability based on ARsets. It is the basis for system redundancy.

CTLSC
This state machine handles the keep alive handling of the CTL established streams.

CTLRSI
This state machine handles the translation between RSI services and RM services.

CIM
The CIM is used to access, based on CMDEV_DA, the CIM ASE of the IO controller in order to read and write the required managed objects. See 5.6.5.

5.6.4.1.2 State transition diagram

The state transition diagram of the CMCTL is shown in Figure 224.

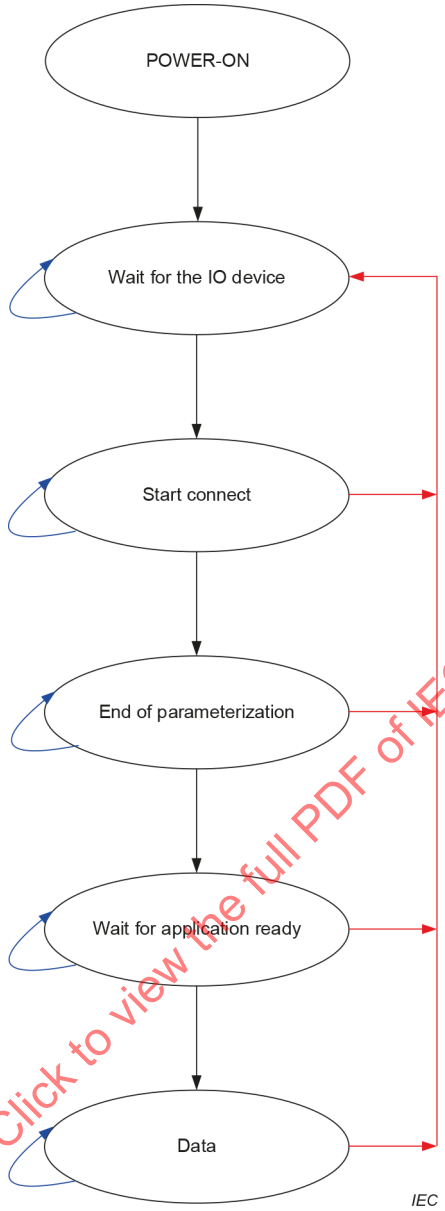


Figure 224 – State transition diagram of the IO controller CM

States of the IO controller CM are:

POWER-ON	Startup the IO controller and initialize all necessary resources
Wait for the IO device	Search for the expected IO device and prepare it, if necessary, for the connection establishing
Start connect	Connect to the IO device by exchanging the communication parameters and the expected submodules to be exchanged in Data
End of parameterization	Convey the parameters for the expected submodules to be exchanged in Data and inform the IO device of the end of the startup parameterization
Wait for application ready	Allow time for the IO device and the parameterized submodules to adapt to the previous conveyed parameters
Data	Cyclic data exchange of the submodules between the IO controller and IO devices

5.6.4.2 Context Management Controller

5.6.4.2.1 Primitive definitions

5.6.4.2.1.1 Primitives exchanged between remote machines

The remote service primitives including their associated parameters issued or received by CMCTL are described in the service definition and shown in Table 1132.

Table 1132 – Remote primitives issued or received by CMCTL

Primitive	Source	Destination	Associated parameters	Functions
RM_Ccontrol.ind	CTLRPC	CMCTL	AREP, ControlBlock, ModuleDiffBlock	—
RM_Connect.cnf (-)	CTLRPC	CMCTL	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Connect.cnf (+)	CTLRPC	CMCTL	AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock, ARRPCBlockRes	—
RM_Dcontrol.cnf (-)	CTLRPC	CMCTL	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Dcontrol.cnf (+)	CTLRPC	CMCTL	AREP, ControlBlock	—
RM_Release.cnf (-)	CTLRPC	CMCTL	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Release.cnf (+)	CTLRPC	CMCTL	AREP, ControlBlock	—
CM_Connect.req	FSPMCTL	CMCTL	AREP, ARBlockReq, ListOfIOCRBlockReq, ListOfExpectedSubmoduleBlockReq, AlarmCRBlockReq, ParameterServerBlock, ListOfMulticastCRBlock, ARRPCBlock, IRInfosBlock, SRInfoBlock, ARVendorBlock	The service Connect opens an application relationship.
RM_Ccontrol.rsp (+)	CMCTL	CTLRPC	AREP, ControlBlock	—
RM_Ccontrol.rsp (-)	CMCTL	CTLRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—

Primitive	Source	Destination	Associated parameters	Functions
RM_Connect.req	CMCTL	CTLRPC	AREP, ARBlockReq, ListOfIOCRBlockReq, AlarmCRBlockReq, ListOfExpectedSubmoduleBlockReq, ARRPCBlockReq	—
RM_Dcontrol.req	CMCTL	CTLRPC	AREP, ControlBlock	—
RM_Release.req	CMCTL	CTLRPC	AREP, ControlBlock	—
CM_Connect.cnf (-)	CMCTL	FSPMCTL	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	This service primitive is a negative confirmation and the requested application relationship is not established.
CM_Connect.cnf (+)	CMCTL	FSPMCTL	AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock, ARRPCBlock	This service primitive is a positive confirmation of the request to establish an application relationship.

5.6.4.2.1.2 Primitives exchanged between local machines

The local service primitives including their associated parameters issued or received by CMCTL are described in the service definition and shown in Table 1133.

Table 1133 – Local primitives issued or received by CMCTL

Primitive	Source	Destination	Associated parameters	Functions
ACCM_req	CMCTL	ACCM	Command={REMOVE}, IPAddress	—
ACCM_cnf	ACCM	CMCTL	Command={REMOVE}	—
CTLDINA_Discover_cnf (-)	CTLDINA	CMCTL	ErrorClass, ErrorCode	—
CTLDINA_Discover_cnf (+)	CTLDINA	CMCTL	—	—
CTLIO_info_ind	CTLIO	CMCTL	AREP	—
CMCTL_start_cnf (-)	CTLSU	CMCTL	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
CMCTL_start_cnf (+)	CTLSU	CMCTL	AREP	—
CMCTL_PrmDone_ind	CTLWRI	CMCTL	AREP, state { PrmEndNeeded, PrmEndDone }	—
CM_Abort_req	FSPMCTL	CMCTL	AREP	The service Abort terminates the AR.
CTLDINA_Discover_req ()	CMCTL	CTLDINA	AddressResolution, StationName, Dev_IP_Parameter, DCP_Parameter	—
CTLSM_start_req	CMCTL	CTLSM	AREP	—
CMCTL_start_req	CMCTL	CTLSU	AREP	—

Primitive	Source	Destination	Associated parameters	Functions
CM_Abort_cnf	CMCTL	FSPMCTL	AREP	—
CMCTL_state_ind	CMCTL	FSPMCTL CTLSU CTLWRI CTLWRR CTLPBE CTLIO CTLRDI CTLSC	AREP, state {STARTUP, APPLRDY, WDATA, DATA, ABORT}	—
NME_AddStream_req	CMCTL	NME	AREP, CREP	This service requests a path in a NME domain.
NME_AddStream_cnf (+)	CMCTL	NME	AREP, CREP, List of StreamIdentification, List of StreamAttributes	This service returns the required path identifiers and the path attributes.
NME_AddStream_cnf (-)	CMCTL	NME	AREP, CREP, Status	—
RSI_I_Add_req	CMCTL	RSII	ISAP, RMAC, VendorID, DeviceID, Instance	—
RSI_I_Add_cnf ()	RSII	CMCTL	ISAP, PNIOStatus	—
SecureChannel- Establishment_req ()	CMCTL	CTLSAM	AREP	Establish a secure channel to protect cyclic and acyclic communication.
SecureChannel- Establishment_cnf ()	CTLSAM	CMCTL	AREP, PNIOStatus	—
CreateLogbookEntry ()	CMCTL	FAL	AREP, PNIOStatus	Create logbook entry.

5.6.4.2.2 State transition diagram

The state transition diagram of the CMCTL is shown in Figure 225.

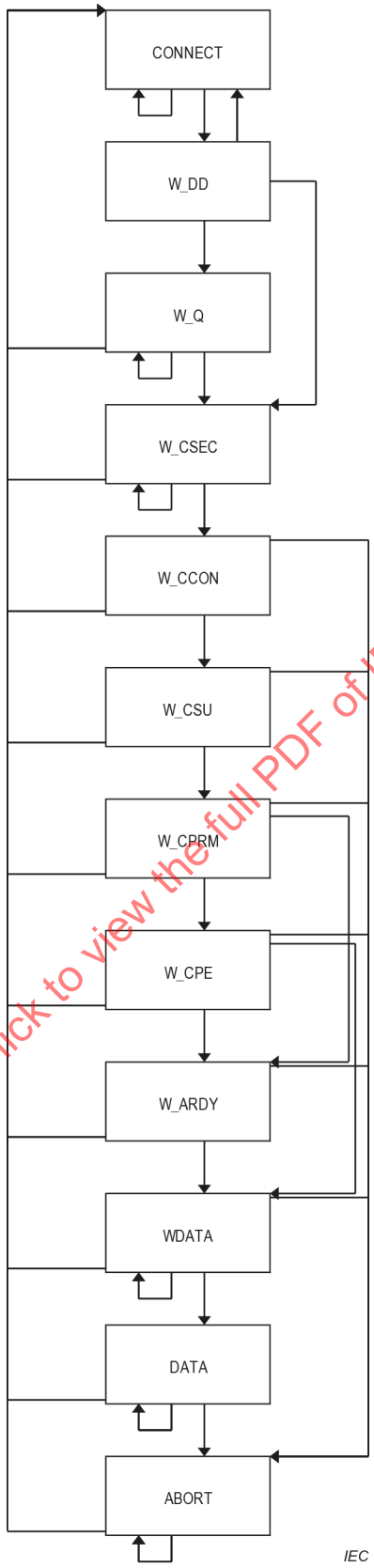


Figure 225 – State transition diagram of CMCTL

States of the CMCTL are:

CONNECT	Wait for a connect service from the FAL
W_DD	Wait for successful device discovery and start the AR
W_Q	Wait for the confirmation of the RSI and NME requests and start the necessary machines
W_CSEC	Wait for confirmation of the secure channel establishment
W_CCON	Wait for the confirmation of the connect request and start the necessary machines
W_CSU	Wait for the confirmation of the start of the necessary machines
W_CPRM	Wait for the end of the startup parameterization to issue the PrmEnd command
W_CPE	Wait for the confirmation of the PrmEnd command and change to the next state
W_ARDY	Wait for the ApplRdy command from the IO device
WDATA	Wait for data exchange using CPM and PPM and the termination of the RPC connect monitoring. The ALPMI is now able to issue an alarm.
DATA	Data exchange and connection monitoring using the CPM / PPM. The ALPMI is now able to issue an alarm.
ABORT	Perform the shutdown in case of application command. An erroneous shutdown is handled by all machines concurrently initiated by the CMCTL_state_ind (ABORT).

5.6.4.2.3 State machine description

The CM Controller protocol machine exists for every AR of an IO controller. The CM Connect request service primitive may be used to address either the endpoint mapper port or the Responder RPC Server port directly. It is necessary to get the port number of the Responder RPC Server port by local means or by querying the responder's endpoint mapper.

An implementation of the CMCTL shall handle the outstanding confirmations in case of an ABORT by itself before a new connect service is issued. Additionally, each ABORT of the connection establishment removes (ACCM_req (Command:=REMOVE, ...)) the stored static ARP cache entry for this AR.

Special case: Dynamic Reconfiguration
The ACCM handling needs to consider that two ARs can be established to the same device and thus, the removal of one of these ARs shall not remove the assigned ACCM entry.

Special case: Device Access
This state machine is used to connect to the CMDEV_DA. In this case this state machine is used in a mode named CMCTL_DA.

NOTE The direct addressing of the Responder Server RPC Port is used to avoid firewall conflicts on PC based systems. The direct addressing of the Initiator Server RPC Port is used to support multiple engineering tools hosted on one PC.

5.6.4.2.4 CMCTL state table

Table 1134 contains the complete description of the CMCTL state machine.

Table 1134 – CMCTL state table

#	Current State	Event /Condition =>Action	Next State
1	CONNECT	**CM_Connect.req ()** => Store Args CTLDINA_Discover_req ()	W_DD
2	CONNECT	**CM_Abort_req ()** => CM_Abort_cnf ()	CONNECT
3	W_DD	**CTLDINA_Discover_cnf (+)** /TRUE == QueryStreamFromNME () => NME_AddStream_req ()	W_Q
4	W_DD	**CTLDINA_Discover_cnf (+)** /FALSE == QueryStreamFromNME () =>	W_CSEC
5	W_DD	**CTLDINA_Discover_cnf (-)** => CM_Connect.cnf (-)	CONNECT
6	W_DD	**CM_Abort_req ()** => CM_Abort_cnf ()	CONNECT
7	W_DD	**CMCTL_state_ind ()** /state == ABORT => ignore	CONNECT
8	W_Q	**RSI_I_Add_cnf (+)** =>	W_CSEC
9	W_Q	**RSI_I_Add_cnf (-)** => CM_Connect.cnf (-)	CONNECT
10	W_Q	**NME_AddStream_cnf (+)** => RSI_I_Add_req ()	W_Q
11	W_Q	**NME_AddStream_cnf (-)** => CM_Connect.cnf (-)	CONNECT
12	W_Q	**CM_Abort_req ()** => CM_Abort_cnf ()	CONNECT
13	W_Q	**CMCTL_state_ind ()** /state == ABORT => ignore	CONNECT
14	W_CSEC	/SecureChannelNeeded => SecureChannelEstablishment_req ()	W_CSEC
15	W_CSEC	/! SecureChannelNeeded => Use stored Args RM_Connect.req ()	W_CCON
16	W_CSEC	**SecureChannelEstablishment_cnf (+)** => Start the associated PRO state machines Use stored Args RM_Connect.req ()	W_CCON

#	Current State	Event /Condition =>Action	Next State
17	W_CSEC	**SecureChannelEstablishment_cnf (-)** => CreateLogBookEntry () CM_Connect.cnf (-)	CONNECT
18	W_CCON	**RM_Connect.cnf (+)** => Store Result if SecureChannelNeeded Start the remaining PRO state machines CMCTL_start_req () CTLSM_start_req ()	W_CSU
19	W_CCON	**RM_Connect.cnf (-)** => CM_Connect.cnf (-)	CONNECT
20	W_CCON	**CM_Abort_req ()** => CMCTL_state_ind (ABORT) RM_Release.req ()	ABORT
21	W_CCON	**CMCTL_state_ind ()** /state == ABORT => ignore	CONNECT
22	W_CSU	**CMCTL_start_cnf (+)** => LocalState := STARTUP CMCTL_state_ind (STARTUP)	W_CPRM
23	W_CSU	**CMCTL_start_cnf (-)** => CMCTL_state_ind (ABORT)	CONNECT
24	W_CSU	**CM_Abort_req ()** => CMCTL_state_ind (ABORT) RM_Release.req ()	ABORT
25	W_CSU	**CMCTL_state_ind ()** /state == ABORT => ignore	CONNECT
26	W_CPRM	**CMCTL_PrmDone_ind ()** /state == PrmEndNeeded => Create PDU ControlBlockConnect.PrmEnd := TRUE RM_Dcontrol.req () //NOTE Alarm are now accepted	W_CPE
27	W_CPRM	**CMCTL_PrmDone_ind ()** /state == PrmEndDone => LocalState := APPLRDY StartTimer (RemoteApplicationReadyTimeout) CMCTL_state_ind (APPLRDY)	W_ARDY
28	W_CPRM	**CM_Abort_req ()** => CMCTL_state_ind (ABORT) RM_Release.req ()	ABORT
29	W_CPRM	**CMCTL_state_ind ()** /state == ABORT => ignore	CONNECT

#	Current State	Event /Condition =>Action	Next State
30	W_CPE	**RM_Dcontrol.cnf (+)** /ControlBlockConnect.PrmEnd == TRUE => LocalState := APPLRDY StartTimer (RemoteApplicationReadyTimeout) CMCTL_state_ind (APPLRDY)	W_ARDY
31	W_CPE	**RM_Dcontrol.cnf (-)** /ControlBlockConnect.PrmEnd == TRUE => CMCTL_state_ind (ABORT)	CONNECT
32	W_CPE	**CM_Abort_req ()** => CMCTL_state_ind (ABORT) RM_Release.req ()	ABORT
33	W_CPE	**RM_Ccontrol.ind ()** /ControlCommand.ApplicationReady == TRUE => Store Result LocalState := WDATA CMCTL_state_ind (WDATA) RM_Ccontrol.rsp (+)	WDATA
34	W_CPE	**CMCTL_state_ind ()** /state == ABORT => ignore	CONNECT
35	W_ARDY	**TimerExpired (RemoteApplicationReadyTimeout)** => CMCTL_state_ind (ABORT)	CONNECT
36	W_ARDY	**RM_Ccontrol.ind ()** /ControlCommand.ApplicationReady == TRUE => Store Result LocalState := WDATA StopTimer (RemoteApplicationReadyTimeout) CMCTL_state_ind (WDATA) RM_Ccontrol.rsp (+)	WDATA
37	W_ARDY	**CM_Abort_req ()** => StopTimer (RemoteApplicationReadyTimeout) CMCTL_state_ind (ABORT) RM_Release.req ()	ABORT
38	W_ARDY	**CMCTL_state_ind ()** /state == ABORT => StopTimer (RemoteApplicationReadyTimeout)	CONNECT
39	WDATA	**RM_Dcontrol.cnf (+)** /ControlBlockConnect.PrmEnd == TRUE => ignore	WDATA
40	WDATA	**RM_Dcontrol.cnf (-)** /ControlBlockConnect.PrmEnd == TRUE => CMCTL_state_ind (ABORT)	CONNECT
41	WDATA	**CM_Abort_req ()** => CMCTL_state_ind (ABORT) RM_Release.req ()	ABORT
42	WDATA	**CTLIO_info_ind ()** /state == DATA_IMPOSSIBLE => ignore	WDATA