

INTERNATIONAL STANDARD



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

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2019 IEC, Geneva, Switzerland

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

IEC Central Office
3, rue de Varembe
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 It also gives information on projects, replaced and withdrawn publications.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.

Electropedia - www.electropedia.org

The world's leading online dictionary on electrotechnology, containing more than 22 000 terminological entries in English and French, with equivalent terms in 16 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary

67 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IECNORM.COM : Click to view the full text of IEC 61858-6:2019

INTERNATIONAL STANDARD



**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 978-2-8322-7010-3

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

CONTENTS

FOREWORD.....	37
INTRODUCTION.....	39
1 Scope.....	41
1.1 General.....	41
1.2 Specifications	41
1.3 Conformance	41
2 Normative references	42
3 Terms, definitions, abbreviated terms, symbols and conventions	45
3.1 Referenced terms and definitions.....	45
3.1.1 ISO/IEC 7498-1 terms.....	45
3.1.2 ISO/IEC 8822 terms.....	45
3.1.3 ISO/IEC 8824-1 terms.....	45
3.1.4 ISO/IEC 9545 terms.....	45
3.2 Terms and definitions for decentralized periphery	46
3.3 Abbreviated terms and symbols	54
3.3.1 Abbreviated terms and symbols for media redundancy	54
3.3.2 Abbreviated terms and symbols for decentralized periphery.....	54
3.3.3 Abbreviated terms and symbols for services	58
3.3.4 Abbreviated terms and symbols for IEEE 802.1Q.....	58
3.3.5 Abbreviated terms and symbols for IETF RFC 2474.....	58
3.3.6 Abbreviated terms and symbols for IETF RFC 4291.....	58
3.4 Conventions.....	58
3.4.1 General concept	58
3.4.2 Conventions for decentralized periphery	58
3.4.3 Conventions used in state machines.....	67
4 Application layer protocol specification for common protocols.....	72
4.1 FAL syntax description.....	72
4.1.1 DLPDU abstract syntax reference	72
4.1.2 Data types.....	74
4.2 Transfer syntax.....	75
4.2.1 Coding of basic data types	75
4.2.2 Coding section related to common basic fields	83
4.3 Discovery and basic configuration.....	94
4.3.1 DCP syntax description	94
4.3.2 DCP protocol state machines.....	122
4.3.3 DLL Mapping Protocol Machines.....	139
4.4 Precision working time control	140
4.4.1 FAL syntax description	140
4.4.2 AP-Context state machine	151
4.4.3 FAL Service Protocol Machines	151
4.4.4 Application Relationship Protocol Machines.....	152
4.4.5 DLL Mapping Protocol Machines.....	215
4.5 Time synchronization	215
4.5.1 General	215
4.5.2 GlobalTime	216
4.5.3 WorkingClock	216
4.6 Media redundancy	217

4.6.1	Media redundancy and loop prevention.....	217
4.6.2	Seamless media redundancy	220
4.7	Real time cyclic.....	220
4.7.1	FAL syntax description	220
4.7.2	FAL transfer syntax	221
4.7.3	FAL Service Protocol Machines	231
4.7.4	Application Relationship Protocol Machines	231
4.7.5	DLL Mapping Protocol Machines.....	249
4.8	Real time acyclic.....	249
4.8.1	RTA syntax description	249
4.8.2	RTA transfer syntax	250
4.8.3	FAL Service Protocol Machines	254
4.8.4	Application Relationship Protocol Machines	254
4.8.5	DLL Mapping Protocol Machines.....	269
4.9	Fragmentation.....	269
4.9.1	General	269
4.9.2	FRAG syntax description	272
4.9.3	FRAG transfer syntax	273
4.9.4	FAL Service Protocol Machines	275
4.9.5	Application Relationship Protocol Machines	275
4.9.6	DLL Mapping Protocol Machines.....	275
4.10	Remote procedure call	286
4.10.1	General	286
4.10.2	RPC syntax description	286
4.10.3	RPC Transfer syntax	288
4.10.4	FAL Service Protocol Machines	304
4.10.5	Application Relationship Protocol Machines	304
4.10.6	DLL Mapping Protocol Machines.....	305
4.11	Link layer discovery	305
4.11.1	General	305
4.11.2	FAL common syntax description	305
4.11.3	LLDP transfer syntax	307
4.11.4	FAL Service Protocol Machines	317
4.11.5	Application Relation Protocol Machines	317
4.11.6	DLL Mapping Protocol Machines.....	317
4.12	Bridges and End Stations.....	317
4.12.1	General	317
4.12.2	Model	318
4.12.3	Traffic Shaping	333
4.12.4	Bridge extensions	334
4.12.5	QueueHandler	335
4.12.6	FAL Service Protocol Machines	335
4.12.7	Application Relation Protocol Machines	335
4.12.8	DLL Mapping Protocol Machines.....	335
4.13	IP suite	374
4.13.1	Overview	374
4.13.2	IP/UDP syntax description	374
4.13.3	IP/UDP transfer syntax	375
4.13.4	ARP.....	378

4.14	Domain name system.....	380
4.14.1	General	380
4.14.2	Primitive definitions	380
4.14.3	DNS state transition diagram	381
4.14.4	State machine description	381
4.14.5	DNS state table	381
4.14.6	Functions, Macros, Timers and Variables	381
4.15	Dynamic host configuration	381
4.15.1	General	381
4.15.2	Primitive definitions	382
4.15.3	DHCP state transition diagram.....	382
4.15.4	State machine description	382
4.15.5	DHCP state table	382
4.15.6	Functions, Macros, Timers and Variables	382
4.16	Simple network management	383
4.16.1	Overview	383
4.16.2	IETF RFC 1213-MIB	383
4.16.3	Enterprise number for PNIO MIB	383
4.16.4	MIB cross reference	384
4.16.5	Behavior in case of modular built bridges	384
4.16.6	LLDP EXT MIB	384
4.17	Common DLL Mapping Protocol Machines	384
4.17.1	Overview	384
4.17.2	Data Link Layer Mapping Protocol Machine	385
4.18	Additional definitions.....	390
5	Application layer protocol specification for decentralized periphery.....	390
5.1	FAL syntax description.....	390
5.1.1	DLPDU abstract syntax reference	390
5.1.2	APDU abstract syntax.....	390
5.2	Transfer syntax.....	409
5.2.1	Coding section related to BlockHeader specific fields	409
5.2.2	Coding section related to RTA-SDU specific fields.....	424
5.2.3	Coding section related to common address fields	429
5.2.4	Coding section related to AL services	445
5.2.5	Coding section related to ARVendorBlock.....	479
5.2.6	Coding section related to PNIOStatus.....	481
5.2.7	Coding section related to I&M Records	498
5.2.8	Coding section related to Alarm and Diagnosis PDUs	505
5.2.9	Coding section related to upload and retrieval	527
5.2.10	Coding section related to iParameter	527
5.2.11	Coding section related to Physical Device Interface Data	528
5.2.12	Coding section related to Physical Device Port Data.....	528
5.2.13	Coding section related to Physical Device IR Data.....	531
5.2.14	Coding section related to Physical Sync Data	554
5.2.15	Coding section related to Isochrone Mode Data	559
5.2.16	Coding section related to Physical Time Data	561
5.2.17	Coding section related to Media Redundancy	564
5.2.18	Coding section related to fiber optics	575
5.2.19	Coding section related to network components	577

5.2.20	Coding section related port statistic	578
5.2.21	Coding section related to fast startup	581
5.2.22	Coding section related to DFP	583
5.2.23	Coding section related to MRPD	587
5.2.24	Coding section related to auto configuration	588
5.2.25	Coding section related to controller to controller communication	591
5.2.26	Coding section related to system redundancy	592
5.2.27	Coding section related to energy saving	595
5.2.28	Coding section related to asset management	595
5.2.29	Coding section related to reporting system	600
5.2.30	Coding section related to Logbook	606
5.2.31	Coding section related to Time	607
5.2.32	Coding section related to Channel Related Process Alarm Reason	607
5.2.33	PDU checking rules	610
5.3	FAL protocol state machines	643
5.3.1	Overall structure	643
5.4	AP-Context state machine	645
5.5	FAL Service Protocol Machines	645
5.5.1	Overview	645
5.5.2	FAL Service Protocol Machine Device	645
5.5.3	FAL Service Protocol Machine Controller	654
5.6	Application Relationship Protocol Machines	665
5.6.1	Alarm Protocol Machine Initiator	665
5.6.2	Alarm Protocol Machine Responder	669
5.6.3	Device	673
5.6.4	Controller	756
5.7	DLL Mapping Protocol Machines	818
Annex A (normative)	Unified establishing of an AR for all RT classes	819
A.1	General	819
A.2	AR establishing	820
A.3	Startup of Alarm transmitter and receiver	825
Annex B (normative)	Compatible establishing of an AR	828
Annex C (informative)	Establishing of a device access AR	831
Annex D (informative)	Establishing of an AR (accelerated procedure)	832
Annex E (informative)	Establishing of an AR (fast startup procedure)	835
Annex F (informative)	Example of the upload, storage and retrieval procedure	837
Annex G (informative)	OSI reference model layers	839
Annex H (informative)	Overview of the IO controller and the IO device state machines	840
Annex I (informative)	Priority regeneration	842
Annex J (informative)	Overview of the PTCP synchronization master hierarchy	843
Annex K (informative)	Optimization of bandwidth usage	845
Annex L (informative)	Time constraints for bandwidth allocation	847
Annex M (informative)	Time constraints for the forwarding of a frame	849
M.1	Principle	849
M.2	Forwarding	849
Annex N (informative)	Principle of dynamic frame packing	851
Annex O (informative)	Principle of Fragmentation	855

Annex P (informative) MRPD – Principle of seamless media redundancy	858
Annex Q (normative) Principle of a RED_RELAY without forwarding information in PDIRFrameData	860
Annex R (informative) Optimization for fast startup without autonegotiation	863
Annex S (informative) Example of a PrmBegin, PrmEnd and ApplRdy sequence	866
Annex T (informative) List of supported MIBs	867
Annex U (informative) Structure and content of BLOB	868
Annex V (normative) LLDP EXT MIB	869
Annex W (normative) Cross reference to the IEC 62439-2	887
W.1 Cross reference to the IEC 62439-2	887
W.1.1 General	887
W.1.2 Ring	887
W.1.3 Interconnection	888
Annex X (normative) Maintaining statistic counters for Ethernet	890
X.1 General	890
X.2 Counting model	890
X.3 Explanation of the IETF RFC defined statistic counters	892
X.4 Value range of the IETF RFC defined statistic counters	893
Bibliography	894
Figure 1 – Common structure of specific fields for octet 1 (high)	60
Figure 2 – Common structure of specific fields for octet 2	60
Figure 3 – Common structure of specific fields for octet 3	60
Figure 4 – Common structure of specific fields for octet 4	61
Figure 5 – Common structure of specific fields for octet 5	61
Figure 6 – Common structure of specific fields for octet 6	61
Figure 7 – Common structure of specific fields for octet 7	62
Figure 8 – Common structure of specific fields for octet 8	62
Figure 9 – Common structure of specific fields for octet 9	62
Figure 10 – Common structure of specific fields for octet 10	63
Figure 11 – Common structure of specific fields for octet 11	63
Figure 12 – Common structure of specific fields for octet 12	63
Figure 13 – Common structure of specific fields for octet 13	64
Figure 14 – Common structure of specific fields for octet 14	64
Figure 15 – Common structure of specific fields for octet 15	64
Figure 16 – Common structure of specific fields for octet 16 (low)	65
Figure 17 – Coding of the data type BinaryDate	77
Figure 18 – Encoding of TimeOfDay with date indication value	77
Figure 19 – Encoding of TimeOfDay without date indication value	78
Figure 20 – Encoding of TimeDifference with date indication value	78
Figure 21 – Encoding of TimeDifference without date indication value	78
Figure 22 – Encoding of a NetworkTime value	79
Figure 23 – Encoding of NetworkTimeDifference value	79
Figure 24 – Encoding of TimeStamp value	80

Figure 25 – Encoding of TimeStampDifference value	81
Figure 26 – Encoding of TimeStampDifferenceShort value	82
Figure 27 – FastForwardingMulticastMACAdd	88
Figure 28 – State transition diagram of DCPUCS	123
Figure 29 – State transition diagram of DCPUCR	127
Figure 30 – State transition diagram of DCPMCS	131
Figure 31 – State transition diagram of DCPMCR	134
Figure 32 – State transition diagram of DCPHMCS	137
Figure 33 – State transition diagram of DCPHMCR	139
Figure 34 – PTCP_SequenceID value range	144
Figure 35 – Timescale correspondence between PTCP_Time and CycleCounter	147
Figure 36 – Message timestamp point	152
Figure 37 – Timer model	152
Figure 38 – Four message timestamps	153
Figure 39 – Line delay protocol with follow up	154
Figure 40 – Line delay protocol without follow up	154
Figure 41 – Line delay measurement	156
Figure 42 – Model parameter for GSDML usage	158
Figure 43 – Bridge delay measurement	159
Figure 44 – Delay accumulation	160
Figure 45 – Worst case accumulated time deviation of synchronization	161
Figure 46 – Signal generation for measurement of deviation	161
Figure 47 – Measurement of deviation	162
Figure 48 – PTCP master sending Sync-Frame without Follow Up-Frame	163
Figure 49 – PTCP master sending Sync-Frame with FollowUp-Frame	163
Figure 50 – !FU Sync Slave Forwarding Sync-Frame	164
Figure 51 – FU Sync Slave Forwarding Sync- and FollowUp-Frame	165
Figure 52 – FU Sync Slave Forwarding Sync- and Generating FollowUp-Frame	166
Figure 53 – Principle of the monitoring of the line delay measurement	167
Figure 54 – State transition diagram of DELAY_REQ	169
Figure 55 – State transition diagram of DELAY_RSP	177
Figure 56 – Overview of PTCP	181
Figure 57 – State transition diagram of SYN_BMA	184
Figure 58 – State transition diagram of SYN_MPSM	193
Figure 59 – State transition diagram of SYN_SPSM	199
Figure 60 – State transition diagram of SYNC_RELAY	206
Figure 61 – State transition diagram of SCHEDULER	212
Figure 62 – GlobalTime timer model	216
Figure 63 – WorkingClock timer model	217
Figure 64 – Media redundancy – Ring	217
Figure 65 – Media redundancy – Interconnection	219
Figure 66 – CycleCounter value range	222
Figure 67 – Structure of the CycleCounter	223

Figure 68 – Optimized CycleCounter setting	224
Figure 69 – SFCRC16 generation rule	228
Figure 70 – SFCycleCounter value range.....	229
Figure 71 – Basic structure of a PPM with frame structure	232
Figure 72 – Basic structure of a PPM with subframe structure.....	233
Figure 73 – State transition diagram of PPM.....	235
Figure 74 – Basic structure of a CPM.....	239
Figure 75 – State transition diagram of CPM.....	241
Figure 76 – Addressing scheme of RTA	251
Figure 77 – Structure of the APM	255
Figure 78 – Structure of the APMS.....	256
Figure 79 – State transition diagram of APMS.....	258
Figure 80 – Structure of the APMR	263
Figure 81 – State transition diagram of APMR	265
Figure 82 – State transition diagram of FRAG_D	276
Figure 83 – State transition diagram of FRAG_S.....	280
Figure 84 – State transition diagram of DEFRAG	283
Figure 85 – DLL Mapping Protocol Machines (DMPM)	317
Figure 86 – Principle traffic flow model of a bridge.....	322
Figure 87 – Principle resource model of a bridge.....	323
Figure 88 – End station – on port bridge – transmit.....	328
Figure 89 – End station – on port bridge – receive	329
Figure 90 – Bridge with End Station.....	330
Figure 91 – Transmit – one port of a bridge	330
Figure 92 – Forwarding process – bridge	331
Figure 93 – Receive – on port of a bridge	331
Figure 94 – Transmit – Management port.....	332
Figure 95 – Receive – Management port.....	333
Figure 96 – State transition diagram of RTC3PSM	339
Figure 97 – State transition diagram for generating events	343
Figure 98 – State transition diagram of RED_RELAY	345
Figure 99 – Scheme of the DFP_RELAY	349
Figure 100 – Scheme of the DFP_RELAY_INBOUND and DFP_RELAY_IN_STORAGE	349
Figure 101 – Scheme of the DFP_RELAY_OUTBOUND.....	350
Figure 102 – State transition diagram of DFP_RELAY	351
Figure 103 – State transition diagram of DFP_RELAY_INBOUND	354
Figure 104 – State transition diagram of DFP_RELAY_IN_STORAGE.....	358
Figure 105 – State transition diagram of DFP_RELAY_OUTBOUND	362
Figure 106 – State transition diagram of MUX.....	366
Figure 107 – State transition diagram of DEMUX	371
Figure 108 – State transition diagram of ACCM	379
Figure 109 – Structuring of the protocol machines within the DMPM (bridge)	385
Figure 110 – State transition diagram of LMPM.....	388

Figure 111 – AlarmSpecifier.SequenceNumber value range	427
Figure 112 – FrameSendOffset vs. duration of a cycle	472
Figure 113 – Severity classification of fault, maintenance and normal	526
Figure 114 – Calculation principle for a cycle	548
Figure 115 – Calculation principle for the minimum YellowTime	549
Figure 116 – Definition of the reserved interval	556
Figure 117 – Toplevel view to the PLL window	559
Figure 118 – Definition of PLL window	559
Figure 119 – Toplevel view to the time PLL window	562
Figure 120 – Definition of time PLL window	563
Figure 121 – Detection of dropped frames – appear	578
Figure 122 – Detection of dropped frames – disappear	578
Figure 123 – Detection of DFP late error – appear and disappear	586
Figure 124 – MediaRedundancyWatchDog expired – appear and disappear	588
Figure 125 – EndPoint1 and Endpoint2 scheme – above and below	593
Figure 126 – EndPoint1 and Endpoint2 scheme – left and right	593
Figure 127 – Relationship among Protocol Machines	643
Figure 128 – State transition diagram of ALPMI	666
Figure 129 – State transition diagram of ALPMR	670
Figure 130 – Scheme of the IO device CM	674
Figure 131 – State transition diagram of the IO device CM	676
Figure 132 – State transition diagram of CMDEV	680
Figure 133 – Scheme of the IO device CM – device access	685
Figure 134 – State transition diagram of CMDEV_DA	687
Figure 135 – State transition diagram of CMSU	691
Figure 136 – State transition diagram of CMIO	696
Figure 137 – State transition diagram of CMRS	699
Figure 138 – State transition diagram of CMWRR	702
Figure 139 – State transition diagram of CMRDR	707
Figure 140 – State transition diagram of CMSM	709
Figure 141 – State transition diagram of CMPBE	713
Figure 142 – State transition diagram of CMDMC	718
Figure 143 – State transition diagram of CMINA	723
Figure 144 – State transition diagram of CMRPC	734
Figure 145 – Intersection and residual amount using different ARUUID.ConfigIDs	740
Figure 146 – Intersection and removed amount using different ARUUID.ConfigIDs	741
Figure 147 – State transition diagram of CMSRL	742
Figure 148 – Single Input and single Output buffer of CMSRL	748
Figure 149 – Dynamic reconfiguration with CMSRL	749
Figure 150 – Alarm queue management of CMSRL	750
Figure 151 – Reporting System management of CMSRL	751
Figure 152 – Primary: Switchover time between two ARs of an ARset	751
Figure 153 – Backup: Switchover time between two ARs of an ARset	752

Figure 154 – State transition diagram of CMSRL_AL	754
Figure 155 – Scheme of the IO controller CM	757
Figure 156 – State transition diagram of the IO controller CM	758
Figure 157 – State transition diagram of CMCTL	762
Figure 158 – State transition diagram of CTLSM	769
Figure 159 – State transition diagram of CTLIO	771
Figure 160 – State transition diagram of CTLRDI	775
Figure 161 – State transition diagram of CTLRDR	778
Figure 162 – State transition diagram of CTLRPC	782
Figure 163 – State transition diagram of CTLSU	787
Figure 164 – State transition diagram of CTLWRI	792
Figure 165 – State transition diagram of CTLWRR	796
Figure 166 – State transition diagram of CTLPBE	799
Figure 167 – State transition diagram of CTLDINA	805
Figure 168 – Automatic NameOfStation assignment	810
Figure 169 – State transition diagram of CTLSRL	812
Figure 170 – Input and Output buffer of CTLSRL	816
Figure 171 – Input and Output buffer with dynamic reconfiguration	816
Figure 172 – Alarm queue management of CTLSRL	817
Figure 173 – Alarm queue management with dynamic reconfiguration	818
Figure A.1 – Establishing of an AR using RT_CLASS_1, RT_CLASS_2 or RT_CLASS_3 (Initial connection monitoring w/o RT)	820
Figure A.2 – Establishing of an AR using RT_CLASS_1, RT_CLASS_2 or RT_CLASS_3 (Connection monitoring with RT)	821
Figure A.3 – Principle of the data evaluation during startup (delayed RED channel establishment)	822
Figure A.4 – Principle of the data evaluation during startup (immediate RED channel establishment)	823
Figure A.5 – Principle of the data evaluation during startup (Special case: Isochronous mode application)	824
Figure A.6 – Startup of Alarm transmitter and receiver without System Redundancy	825
Figure A.7 – Startup of Alarm transmitter and receiver with System Redundancy	826
Figure A.8 – Startup of Alarm transmitter and receiver during a PrmBegin / PrmEnd / ApplRdy sequence	827
Figure B.1 – Establishing of an AR using RT_CLASS_3 AR with startup mode “Legacy”	829
Figure B.2 – Establishing of an AR using RT_CLASS_1, 2 or UDP AR with startup mode “Legacy”	830
Figure C.1 – Establishing of a device access AR	831
Figure D.1 – Accelerated establishing of an IOAR without error	833
Figure D.2 – Accelerated establishing of an IOAR with “late” error	834
Figure E.1 – Establishing of an IOAR using fast startup	836
Figure F.1 – Example of upload with storage	837
Figure F.2 – Example of retrieval with storage	838
Figure G.1 – Assignment of the OSI reference model layers	839
Figure H.1 – Overview of the IO controller state machines	840

Figure H.2 – Overview of the IO device state machines	840
Figure J.1 – Level model for synchronization master hierarchy	843
Figure J.2 – Two level variant of the synchronization master hierarchy	844
Figure K.1 – Devices build up in a linear structure	845
Figure K.2 – Propagation of frames in linear transmit direction	845
Figure K.3 – Propagation of a frames in receive direction	846
Figure L.1 – Overview of time constraints for bandwidth allocation.....	847
Figure L.2 – Calculation of the length of a RED period.....	847
Figure L.3 – Calculation of the length of a GREEN period	848
Figure M.1 – IEEE 802.3 definition.....	849
Figure M.2 – Minimization of bridge delay.....	849
Figure N.1 – Dynamic frame packing	851
Figure N.2 – Dynamic frame packing – truncation of outputs.....	852
Figure N.3 – Dynamic frame packing – concatenation of inputs.....	852
Figure N.4 – End node mode	853
Figure N.5 – DFPFeed definition.....	853
Figure O.1 – Principle of fragmentation.....	855
Figure O.2 – Protocol elements of fragments	855
Figure O.3 – Bandwidth allocation using fragmentation.....	856
Figure O.4 – Guardian for a fragmentation domain.....	856
Figure P.1 – Principle of seamless media redundancy – I/OCR.....	858
Figure P.2 – Principle of seamless media redundancy – MCR.....	859
Figure P.3 – Principle of seamless media redundancy – Line.....	859
Figure Q.1 – Generating the FrameSendOffset for a RED_RELAY without forwarding information in PDIRFrameData	860
Figure R.1 – Scheme of a 2-port switch	863
Figure R.2 – Scheme of 2-ports	863
Figure S.1 – PrmBegin, PrmEnd and ApplRdy procedure	866
Figure X.1 – IEEE 802 structure used for statistic counters.....	891
Figure X.2 – IEEE 802 summary for statistic counters.....	892
Table 1 – One octet	65
Table 2 – Two subsequent octets.....	66
Table 3 – Four subsequent octets	66
Table 4 – Eight subsequent octets	67
Table 5 – Sixteen subsequent octets	67
Table 6 – State machine description elements	68
Table 7 – Description of state machine elements	68
Table 8 – Conventions used in state machines	69
Table 9 – Conventions for services used in state machines	70
Table 10 – IEEE 802.3 DLPDU syntax	72
Table 11 – IEEE 802.11 DLPDU syntax	73
Table 12 – IEEE 802.15.1 DLPDU syntax	74

Table 13 – Status	79
Table 14 – Time source	81
Table 15 – SourceAddress.....	83
Table 16 – Single port device.....	83
Table 17 – DCP_MulticastMACAdd for Identify	84
Table 18 – DCP_MulticastMACAdd for Hello.....	84
Table 19 – DCP_MulticastMACAdd	84
Table 20 – MulticastMACAdd range 1	84
Table 21 – MulticastMACAdd range 2	84
Table 22 – MulticastMACAdd range 3	85
Table 23 – PTCP_MulticastMACAdd range 2	85
Table 24 – PTCP_MulticastMACAdd range 3	85
Table 25 – PTCP_MulticastMACAdd range 4	85
Table 26 – PTCP_MulticastMACAdd range 5	86
Table 27 – PTCP_MulticastMACAdd range 6	86
Table 28 – PTCP_MulticastMACAdd range 7	86
Table 29 – MulticastMACAdd range 8	86
Table 30 – MulticastMACAdd range 9	86
Table 31 – MulticastMACAdd range 10	87
Table 32 – MulticastMACAdd range 11	87
Table 33 – RT_CLASS_3 destination multicast address	88
Table 34 – RT_CLASS_3 invalid frame multicast address	89
Table 35 – LT (Length/Type).....	89
Table 36 – TagControlInformation.VID	89
Table 37 – TagControlInformation.DEI	90
Table 38 – TagControlInformation.PCP	90
Table 39 – FrameID range 1	90
Table 40 – FrameID range 2	91
Table 41 – FrameID range 3	91
Table 42 – FrameID range 4	91
Table 43 – FrameID range 5	91
Table 44 – FrameID range 6	92
Table 45 – FrameID range 7	92
Table 46 – FrameID range 8	92
Table 47 – FrameID range 9	93
Table 48 – FrameID range 10	93
Table 49 – FrameID range 11	93
Table 50 – FrameID range 12	93
Table 51 – FrameID range 13	93
Table 52 – FragmentationFrameID.FragSequence	94
Table 53 – FragmentationFrameID.Constant.....	94
Table 54 – DCP APDU syntax.....	94
Table 55 – DCP substitutions.....	95

Table 56 – ServiceID	98
Table 57 – ServiceType.Selection	98
Table 58 – ServiceType.Reserved	98
Table 59 – ServiceType.Selection	99
Table 60 – ServiceType.Reserved_1	99
Table 61 – ServiceType.Response	99
Table 62 – ServiceType.Reserved_2	99
Table 63 – ResponseDelayFactor	100
Table 64 – List of options	101
Table 65 – List of suboptions for option IPOption	102
Table 66 – List of suboptions for option DevicePropertiesOption	102
Table 67 – List of suboptions for option DHCPOption	102
Table 68 – List of suboptions for option ControlOption	103
Table 69 – List of suboptions for option DeviceInitiativeOption	103
Table 70 – List of suboptions for option AllSelectorOption	103
Table 71 – List of suboptions for option ManufacturerSpecificOption	103
Table 72 – SuboptionDHCP	105
Table 73 – Coding of DCPBlockLength in conjunction with SuboptionStart	106
Table 74 – Coding of DCPBlockLength in conjunction with SuboptionStop	106
Table 75 – Coding of DCPBlockLength in conjunction with SuboptionSignal	106
Table 76 – Coding of DCPBlockLength in conjunction with SuboptionFactoryReset	107
Table 77 – Alignment between FactoryReset and ResetToFactory	107
Table 78 – Coding of DCPBlockLength in conjunction with SuboptionResetToFactory	107
Table 79 – Meaning of the different ResetToFactory modes	108
Table 80 – Coding of DCPBlockLength in conjunction with SuboptionDeviceInitiative	108
Table 81 – Coding of DCPBlockLength	109
Table 82 – BlockQualifier with options IPOption, DevicePropertiesOption, DHCPOption and ManufacturerSpecificOption	109
Table 83 – BlockQualifier with option ControlOption and suboption SuboptionResetToFactory	110
Table 84 – BlockQualifier with option ControlOption and NOT suboption SuboptionResetToFactory	111
Table 85 – BlockError	111
Table 86 – BlockInfo for SuboptionIPParameter	111
Table 87 – Bit 1 and Bit 0 of BlockInfo for SuboptionIPParameter	112
Table 88 – Bit 7 of BlockInfo for SuboptionIPParameter	112
Table 89 – BlockInfo for all other suboptions	112
Table 90 – DeviceInitiativeValue	112
Table 91 – SignalValue	113
Table 92 – DeviceRoleDetails	115
Table 93 – IPAddress	115
Table 94 – Subnetmask	117
Table 95 – StandardGateway	118
Table 96 – Correlation between the subfields of IPSuite	119

Table 97 – MACAddress as client identifier	120
Table 98 – NameOfStation as client identifier	120
Table 99 – Arbitrary client identifier	120
Table 100 – DHCPParameterValue using DHCP Option 255	121
Table 101 – StandardGatewayValue.StandardGateway	122
Table 102 – Remote primitives issued or received by DCPUCS	122
Table 103 – Local primitives issued or received by DCPUCS	123
Table 104 – DCPUCS state table	124
Table 105 – Functions, Macros, Timers and Variables used by the DCPUCS	126
Table 106 – Remote primitives issued or received by DCPUCR	127
Table 107 – Local primitives issued or received by DCPUCR	127
Table 108 – DCPUCR state table	128
Table 109 – Functions, Macros, Timers and Variables used by the DCPUCR	130
Table 110 – Remote primitives issued or received by DCPMCS	130
Table 111 – Local primitives issued or received by DCPMCS	131
Table 112 – DCPMCS state table	132
Table 113 – Functions used by the DCPMCS	133
Table 114 – Remote primitives issued or received by DCPMCR	134
Table 115 – Local primitives issued or received by DCPMCR	134
Table 116 – DCPMCR state table	135
Table 117 – Functions, Macros, Timers and Variables used by the DCPMCR	136
Table 118 – Remote primitives issued or received by DCPHMCS	136
Table 119 – Local primitives issued or received by DCPHMCS	137
Table 120 – DCPHMCS state table	137
Table 121 – Functions, Macros, Timers and Variables used by the DCPHMCS	138
Table 122 – Remote primitives issued or received by DCPHMCR	138
Table 123 – Local primitives issued or received by DCPHMCR	138
Table 124 – DCPHMCR state table	139
Table 125 – Functions, Macros, Timers and Variables used by the DCPHMCR	139
Table 126 – PTCP APDU syntax	140
Table 127 – PTCP substitutions	140
Table 128 – PTCP_TLVHeader.Type	141
Table 129 – PTCP_Delay10ns	142
Table 130 – PTCP_Delay1ns_Byte.Value	142
Table 131 – PTCP_Delay1ns	143
Table 132 – PTCP_Delay1ns_FUP	143
Table 133 – PTCP_SequenceID	143
Table 134 – PTCP_SubType for OUI (=00-0E-CF)	144
Table 135 – PTCP_Seconds	145
Table 136 – PTCP_NanoSeconds	145
Table 137 – PTCP_Flags.LeapSecond	145
Table 138 – Timescale correspondence between PTCP_EpochNumber, PTCP_Second, PTCP_Nanosecond, CycleCounter and SendClockFactor	146

Table 139 – PTCP_CurrentUTCOffset.....	148
Table 140 – PTCP_MasterPriority1.Priority for SyncID == 0 and SyncProperties.Role == 2	148
Table 141 – PTCP_MasterPriority1.Priority for SyncID == 0 and SyncProperties.Role == 1	148
Table 142 – PTCP_MasterPriority1.Level.....	149
Table 143 – PTCP_MasterPriority2	149
Table 144 – PTCP_ClockClass for SyncID == 0 (working clock synchronization)	149
Table 145 – PTCP_ClockAccuracy.....	150
Table 146 – PTCP_ClockVariance	151
Table 147 – PTCP_T2PortRxDelay	151
Table 148 – PTCP_T3PortTxDelay	151
Table 149 – PTCP_T2TimeStamp	151
Table 150 – Remote primitives issued or received by DELAY_REQ	168
Table 151 – Local primitives issued or received by DELAY_REQ	168
Table 152 – DELAY_REQ state table	170
Table 153 – Functions, macros, timers and variables used by the DELAY_REQ	174
Table 154 – Remote primitives issued or received by DELAY_RSP	176
Table 155 – Local primitives issued or received by DELAY_RSP	176
Table 156 – DELAY_RSP state table	178
Table 157 – Functions, Macros, Timers and Variables used by the DELAY_RSP	180
Table 158 – Remote primitives issued or received by SYN_BMA	182
Table 159 – Local primitives issued or received by SYN_BMA	182
Table 160 – SYN_BMA state table	185
Table 161 – Functions, Macros, Timers and Variables used by the SYN_BMA	189
Table 162 – Remote primitives issued or received by SYN_MPSM	192
Table 163 – Local primitives issued or received by SYN_MPSM	192
Table 164 – SYN_MPSM state table	194
Table 165 – Functions, Macros, Timers and Variables used by the SYN_MPSM	197
Table 166 – Remote primitives issued or received by SYN_SPSM	198
Table 167 – Local primitives issued or received by SYN_SPSM	198
Table 168 – SYN_SPSM state table	200
Table 169 – Functions, Macros, Timers and Variables used by the SYN_SPSM	203
Table 170 – Truth table for one SyncID for receiving sync and follow up frames	204
Table 171 – Remote primitives issued or received by SYNC_RELAY	205
Table 172 – Local primitives issued or received by SYNC_RELAY	205
Table 173 – SYNC_RELAY state table	207
Table 174 – Functions, Macros, Timers and Variables used by the SYNC_RELAY	208
Table 175 – Truth table for one SyncID for receiving	210
Table 176 – Truth table for one SyncID for transmitting	211
Table 177 – Remote primitives issued or received by SCHEDULER	211
Table 178 – Local primitives issued or received by SCHEDULER	212
Table 179 – SCHEDULER state table	213
Table 180 – Functions, Macros, Timers and Variables used by the SCHEDULER	214
Table 181 – Truth table for RxPeriodChecker of one port	215

Table 182 – Truth table for TxPeriodChecker of one port	215
Table 183 – Timescales	215
Table 184 – Timescale correspondence between GlobalTime, TAI and UTC	216
Table 185 – Conjunction between supported MRP_Role and default MRP_Prio	218
Table 186 – Extended forwarding rule	218
Table 187 – Managed Multicast MAC address	219
Table 188 – RTC APDU syntax	220
Table 189 – RTC substitutions	221
Table 190 – CycleCounter Difference	222
Table 191 – DataStatus.State	224
Table 192 – DataStatus.Redundancy in conjunction with DataStatus.State==Backup	225
Table 193 – DataStatus.Redundancy in conjunction with DataStatus.State==Primary	225
Table 194 – DataStatus.DataValid	225
Table 195 – DataStatus.ProviderState	225
Table 196 – DataStatus.StationProblemIndicator	226
Table 197 – DataStatus.Ignore of a frame	226
Table 198 – DataStatus.Ignore of a sub frame	226
Table 199 – TransferStatus for RT_CLASS_3	227
Table 200 – SFPosition.Position	228
Table 201 – SFPosition.Reserved	228
Table 202 – SFDataLength	228
Table 203 – SFCycleCounter Difference	230
Table 204 – IOxS.Extension	230
Table 205 – IOxS.Instance	230
Table 206 – IOxS.DataState	231
Table 207 – APDU_Status of a PPM with subframe structure	233
Table 208 – Remote primitives issued or received by PPM	234
Table 209 – Local primitives issued or received by PPM	234
Table 210 – PPM state table	236
Table 211 – Functions, Macros, Timers and Variables used by the PPM	237
Table 212 – Truth table used by the PPM for TxOption	238
Table 213 – Remote primitives issued or received by CPM	240
Table 214 – Local primitives issued or received by CPM	240
Table 215 – CPM state table	242
Table 216 – Functions, Macros, Timers and Variables used by the CPM	245
Table 217 – Truth table used by the CPM for RxOption	246
Table 218 – Truth table for one frame using RT_CLASS_x	247
Table 219 – Truth table for one frame using RT_CLASS_UDP	247
Table 220 – Truth table for the C_SDU	247
Table 221 – Truth table for arranging DHt and data	248
Table 222 – Truth table for the subframe – frame check	248
Table 223 – Truth table for the subframe – sub frame check	248
Table 224 – Truth table for the subframe – sub frame data check	249

Table 225 – Truth table for the subframe – DHT and data	249
Table 226 – RTA APDU syntax	249
Table 227 – RTA substitutions	250
Table 228 – PDUType.Type	252
Table 229 – PDUType.Version	252
Table 230 – AddFlags.WindowSize	252
Table 231 – AddFlags.TACK	252
Table 232 – SendSeqNum	253
Table 233 – SendSeqNum and AckSeqNum start sequence	253
Table 234 – AckSeqNum	254
Table 235 – VarPartLen	254
Table 236 – Remote primitives issued or received by APMS	256
Table 237 – Local primitives issued or received by APMS	257
Table 238 – APMS state table	259
Table 239 – Functions, Macros, Timers and Variables used by the APMS	261
Table 240 – Remote primitives issued or received by APMR	264
Table 241 – Local primitives issued or received by APMR	265
Table 242 – APMR state table	266
Table 243 – Functions, Macros, Timers and Variables used by the APMR	268
Table 244 – TagControllInformation.PCP vs. streams	269
Table 245 – Lower limit of fragments	272
Table 246 – FRAG APDU syntax	272
Table 247 – FRAG substitutions	273
Table 248 – FragDataLength	274
Table 249 – FragStatus.FragmentNumber	274
Table 250 – FragStatus.Reserved	274
Table 251 – FragStatus.MoreFollows	275
Table 252 – Remote primitives issued or received by FRAG_D	275
Table 253 – Local primitives issued or received by FRAG_D	275
Table 254 – FRAG_D state table (dynamic)	277
Table 255 – Functions, Macros, Timers and Variables used by the FRAG_D (dynamic)	278
Table 256 – Remote primitives issued or received by FRAG_S	279
Table 257 – Local primitives issued or received by FRAG_S	279
Table 258 – FRAG_S state table (static)	281
Table 259 – Functions, Macros, Timers and Variables used by the FRAG_S (static)	282
Table 260 – Remote primitives issued or received by DEFRAG	283
Table 261 – Local primitives issued or received by DEFRAG	283
Table 262 – DEFRAG state table	284
Table 263 – Functions, Macros, Timers and Variables used by the DEFRAG	285
Table 264 – Truth table for the DefragGuard – first fragment	285
Table 265 – Truth table for the DefragGuard – next fragment	285
Table 266 – Truth table for the DefragGuard – last fragment	286
Table 267 – RPC APDU syntax	286

Table 268 – RPC substitutions	287
Table 269 – RPCVersion	288
Table 270 – RPCPacketType	288
Table 271 – RPCFlags	289
Table 272 – RPCFlags2	289
Table 273 – RPCDRep.Character- and IntegerEncoding	290
Table 274 – RPCDRep Octet 2 – Floating Point Representation	290
Table 275 – RPCObjectUUID.Data4	291
Table 276 – RPCObjectUUID for devices	291
Table 277 – Instance or node number	291
Table 278 – RPCInterfaceUUID for PNIO	292
Table 279 – RPCInterfaceUUID for the RPC end point mapper	292
Table 280 – RPCInterfaceVersion.Major	293
Table 281 – RPCInterfaceVersion.Minor	293
Table 282 – RPCOperationNmb (IO device, controller and supervisor)	294
Table 283 – RPCOperationNmb for endpoint mapper	294
Table 284 – RPCVersionFack	295
Table 285 – RPCDataRepresentationUUID – defined values	296
Table 286 – RPCInquiryType	298
Table 287 – RPCEPMapStatus	300
Table 288 – Values of NCAFaultStatus	302
Table 289 – Values of NCARrejectStatus	303
Table 290 – Remote primitives issued or received by RPC	304
Table 291 – Local primitives issued or received by RPC	304
Table 292 – LLDP APDU syntax	306
Table 293 – LLDP substitutions	306
Table 294 – LLDP_ChassisID in conjunction with MultipleInterfaceMode.NameOfDevice == 0 and NameOfStation	307
Table 295 – LLDP_ChassisID in conjunction with MultipleInterfaceMode.NameOfDevice == 1	307
Table 296 – LLDP_PortID in conjunction with MultipleInterfaceMode.NameOfDevice	308
Table 297 – LLDP_PNIO_SubType	308
Table 298 – PTCP_PortRxDelayLocal	309
Table 299 – PTCP_PortRxDelayRemote	309
Table 300 – PTCP_PortTxDelayLocal	309
Table 301 – PTCP_PortTxDelayRemote	309
Table 302 – CableDelayLocal	310
Table 303 – RTClass2_PortStatus.State with ARProperties.StartupMode == Legacy	310
Table 304 – RTClass2_PortStatus.State with ARProperties.StartupMode == Advanced	310
Table 305 – RTClass3_PortStatus.State	311
Table 306 – RTClass3_PortStatus.Fragmentation	311
Table 307 – RTClass3_PortStatus.PreambleLength	311
Table 308 – Truth table for shortening of the preamble	312
Table 309 – RTClass3_PortStatus.Optimized	312

Table 310 – MRRT_PortStatus.State	313
Table 311 – IRDataUUID	313
Table 312 – LLDP_RedOrangePeriodBegin.Offset	313
Table 313 – LLDP_RedOrangePeriodBegin.Valid	313
Table 314 – LLDP_OrangePeriodBegin.Offset	314
Table 315 – LLDP_OrangePeriodBegin.Valid with ARProperties.StartupMode == Legacy	314
Table 316 – LLDP_OrangePeriodBegin.Valid with ARProperties.StartupMode == Advanced	314
Table 317 – LLDP_GreenPeriodBegin.Offset	315
Table 318 – LLDP_GreenPeriodBegin.Valid	315
Table 319 – LLDP_LengthOfPeriod.Length	315
Table 320 – LLDP_LengthOfPeriod.Valid	315
Table 321 – Priority remapping at an ingress boundary port	319
Table 322 – Trees and FDBs	320
Table 323 – Available queue	322
Table 324 – Queue related memory management	323
Table 325 – Queue usage	324
Table 326 – Queue usage	324
Table 327 – QB TSA usage	325
Table 328 – QB TSA usage	326
Table 329 – Traffic Classes[0..7] for eight queues	334
Table 330 – Traffic Classes[0..3] for four queues	334
Table 331 – Unicast FDB entries	335
Table 332 – Multicast FDB entries	336
Table 333 – Broadcast FDB entry	337
Table 334 – Remote primitives issued or received by MAC_RELAY	337
Table 335 – Local primitives issued or received by MAC_RELAY	338
Table 336 – Functions, Macros, Timers and Variables used by the MAC_RELAY	338
Table 337 – Remote primitives issued or received by RTC3PSM	339
Table 338 – Local primitives issued or received by RTC3PSM	339
Table 339 – RTC3PSM state table	340
Table 340 – Functions, Macros, Timers and Variables used by the RTC3PSM	341
Table 341 – Truth table for the RTC3PSM	342
Table 342 – RXBeginEndAssignment and TXBeginEndAssignment	342
Table 343 – Event function table	343
Table 344 – Remote primitives issued or received by RED_RELAY	344
Table 345 – Local primitives issued or received by RED_RELAY	344
Table 346 – RED_RELAY state table	346
Table 347 – Functions, Macros, Timers and Variables used by the RED_RELAY	347
Table 348 – Truth table for the RedGuard with full check	347
Table 349 – Truth table for the RedGuard with reduced check	348
Table 350 – Truth table for the RedGuard with minimal check	348
Table 351 – Remote primitives issued or received by DFP_RELAY	350

Table 352 – Local primitives issued or received by DFP_RELAY	351
Table 353 – DFP_RELAY state table	352
Table 354 – Functions, Macros, Timers and Variables used by the DFP_RELAY	352
Table 355 – Truth table for the DFPGuard	353
Table 356 – Remote primitives issued or received by DFP_RELAY_INBOUND	353
Table 357 – Local primitives issued or received by DFP_RELAY_INBOUND	354
Table 358 – DFP_RELAY_INBOUND state table	355
Table 359 – Functions, Macros, Timers and Variables used by the DFP_RELAY_INBOUND	355
Table 360 – Truth table for the InboundGuard – frame check	356
Table 361 – Truth table for the InboundGuard – sub frame check	356
Table 362 – Truth table for the InboundGuard – sub frame data check	356
Table 363 – Truth table for the InboundGuard – full check	357
Table 364 – Remote primitives issued or received by DFP_RELAY_IN_STORAGE	357
Table 365 – Local primitives issued or received by DFP_RELAY_IN_STORAGE	358
Table 366 – DFP_RELAY_IN_STORAGE state table	359
Table 367 – Functions, Macros, Timers and Variables used by the DFP_RELAY_IN_STORAGE	360
Table 368 – Remote primitives issued or received by DFP_RELAY_OUTBOUND	361
Table 369 – Local primitives issued or received by DFP_RELAY_OUTBOUND	361
Table 370 – APDU_Status used if frame is shortened	362
Table 371 – DFP_RELAY_OUTBOUND state table	363
Table 372 – Functions, Macros, Timers and Variables used by the DFP_RELAY_OUTBOUND	364
Table 373 – Truth table for the OutboundGuard – frame check	364
Table 374 – Truth table for the OutboundGuard – sub frame check	365
Table 375 – Remote primitives issued or received by MUX	365
Table 376 – Local primitives issued or received by MUX	365
Table 377 – MUX state table	367
Table 378 – Functions, Macros, Timers and Variables used by MUX	368
Table 379 – Truth table for FrameSizeFits	369
Table 380 – Truth table for StateChecker	369
Table 381 – Remote primitives issued or received by DEMUX	370
Table 382 – Local primitives issued or received by DEMUX	370
Table 383 – DEMUX state table	372
Table 384 – Functions, Macros, Timers and Variables used by the DEMUX	373
Table 385 – IP/UDP APDU syntax	374
Table 386 – IP/UDP substitutions	375
Table 387 – UDP_SrcPort	376
Table 388 – UDP_DstPort	376
Table 389 – IP_DstIPAddress	376
Table 390 – IP Multicast DstIPAddress according to IETF RFC 2365	376
Table 391 – IP_DifferentiatedServices.DSCP	377
Table 392 – IP_DifferentiatedServices.ECN	377

Table 393 – Remote primitives issued or received by ACCM	378
Table 394 – Local primitives issued or received by ACCM	379
Table 395 – ACCM state table	380
Table 396 – Functions, Macros, Timers and Variables used by the ACCM	380
Table 397 – Remote primitives issued or received by DNS	381
Table 398 – Local primitives issued or received by DNS	381
Table 399 – Functions, Macros, Timers and Variables used by the DNS	381
Table 400 – Remote primitives issued or received by DHCP	382
Table 401 – Local primitives issued or received by machines	382
Table 402 – Functions, Macros, Timers and Variables used by the DHCP	383
Table 403 – List of supported IETF RFC 1213-MIB objects	383
Table 404 – Enterprise number	384
Table 405 – Cross reference – MIBs	384
Table 406 – Cross reference – PDPortDataAdjust	384
Table 407 – Remote primitives issued or received by LMPM	386
Table 408 – Local primitives issued or received by LMPM	387
Table 409 – LMPM state table	388
Table 410 – Functions, Macros, Timers and Variables used by the LMPM	389
Table 411 – IO APDU substitutions	391
Table 412 – BlockType	409
Table 413 – BlockLength	422
Table 414 – BlockVersionHigh	423
Table 415 – BlockVersionLow	423
Table 416 – AlarmType	424
Table 417 – AlarmSpecifier.SequenceNumber	427
Table 418 – AlarmSpecifier.SequenceNumber Difference	428
Table 419 – AlarmSpecifier.ChannelDiagnosis	428
Table 420 – AlarmSpecifier.ManufacturerSpecificDiagnosis	428
Table 421 – AlarmSpecifier.SubmoduleDiagnosisState	429
Table 422 – AlarmSpecifier.ARDiagnosisState	429
Table 423 – API	430
Table 424 – SlotNumber	430
Table 425 – SubslotNumber	430
Table 426 – Index range	431
Table 427 – Expression 1 (subslot specific)	432
Table 428 – Expression 2 (slot specific)	432
Table 429 – Expression 3 (AR specific)	432
Table 430 – Expression 4 (API specific)	432
Table 431 – Expression 5 (device specific)	432
Table 432 – Grouping of DiagnosisData	433
Table 433 – Index (user specific)	434
Table 434 – Index (subslot specific)	434
Table 435 – Index (slot specific)	438

Table 436 – Index (AR specific)	439
Table 437 – Index (API specific)	441
Table 438 – Index (device specific)	442
Table 439 – RecordDataLength	445
Table 440 – ARType	445
Table 441 – IOCRMulticastMACAdd using RT_CLASS_UDP	446
Table 442 – IOCRMulticastMACAdd using RT_CLASS_x	446
Table 443 – Type 10 OUI	447
Table 444 – ARProperties.State	447
Table 445 – ARProperties.SupervisorTakeoverAllowed	447
Table 446 – ARProperties.ParameterizationServer	448
Table 447 – ARProperties.DeviceAccess	448
Table 448 – ARProperties.CompanionAR	448
Table 449 – ARProperties.AcknowledgeCompanionAR	448
Table 450 – ARProperties.CombinedObjectContainer with ARProperties.StartupMode == Legacy	449
Table 451 – ARProperties.CombinedObjectContainer with ARProperties.StartupMode == Advanced	449
Table 452 – ARProperties.StartupMode	449
Table 453 – ARProperties.PullModuleAlarmAllowed	449
Table 454 – IOCRProperties.RTClass	450
Table 455 – IOCRTagHeader.IOCRVLANID	451
Table 456 – IOCRTagHeader.IOUserPriority	451
Table 457 – IOCRType	451
Table 458 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess==0	451
Table 459 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess==1 or ARProperties.StartupMode==1	452
Table 460 – CMInitiatorTriggerTimeoutFactor	452
Table 461 – IODataObjectFrameOffset	453
Table 462 – IOCSFrameOffset	453
Table 463 – LengthIOCS	454
Table 464 – LengthIOPS	454
Table 465 – LengthData	454
Table 466 – AlarmCRProperties.Priority	455
Table 467 – AlarmCRProperties.Transport	455
Table 468 – AlarmCRTagHeaderHigh.AlarmCRVLANID	455
Table 469 – AlarmCRTagHeaderHigh.AlarmUserPriority	456
Table 470 – AlarmCRTagHeaderLow.AlarmCRVLANID	456
Table 471 – AlarmCRTagHeaderLow.AlarmUserPriority	456
Table 472 – AlarmSequenceNumber	456
Table 473 – AlarmCRType	457
Table 474 – RTATimeoutFactor	457
Table 475 – RTARetries	457
Table 476 – AddressResolutionProperties.Protocol	458

Table 477 – AddressResolutionProperties.Factor	458
Table 478 – MCITimeoutFactor	459
Table 479 – DeviceIDLow and DeviceIDHigh	459
Table 480 – VendorIDLow	460
Table 481 – VendorIDHigh	460
Table 482 – ModuleIdentNumber	460
Table 483 – SubmoduleIdentNumber	461
Table 484 – ARUUID	462
Table 485 – ARUUID in conjunction with ARTYPE==LOCARSR	462
Table 486 – Conjunction between ARUUID.ARnumber and Endpoint1 or Endpoint2	462
Table 487 – ARUUID.ConfigID generation rule	463
Table 488 – TargetARUUID	463
Table 489 – AdditionalValue1 and AdditionalValue2	463
Table 490 – ControlBlockProperties in conjunction with ControlCommand.ApplicationReady with ARProperties.StartupMode==1	463
Table 491 – ControlBlockProperties in conjunction with ControlCommand.ApplicationReady with ARProperties.StartupMode==0	464
Table 492 – ControlBlockProperties in conjunction with the other values of the field ControlCommand	464
Table 493 – ControlCommand.PrmEnd	464
Table 494 – ControlCommand.ApplicationReady	464
Table 495 – ControlCommand.Release	465
Table 496 – ControlCommand.Done	465
Table 497 – ControlCommand.ReadyForCompanion	465
Table 498 – ControlCommand.ReadyForRT_CLASS_3	465
Table 499 – ControlCommand.PrmBegin	465
Table 500 – DataDescription.Type	466
Table 501 – Values of DataLength	466
Table 502 – Values of SendClockFactor	467
Table 503 – Values of ReductionRatio for RT_CLASS_1 and RT_CLASS_2	468
Table 504 – Values of ReductionRatio for RT_CLASS_3 and SendClockFactor ≥ 8	468
Table 505 – Values of ReductionRatio for RT_CLASS_3 and SendClockFactor < 8	468
Table 506 – Values of ReductionRatio in conjunction with a non power of 2 SendClockFactor	468
Table 507 – Values of ReductionRatio for RT_CLASS_UDP	469
Table 508 – Values of Phase	469
Table 509 – Values of Sequence	469
Table 510 – DataHoldFactor of a frame	470
Table 511 – DataHoldFactor of a Subframe	470
Table 512 – Values of FrameSendOffset	471
Table 513 – ModuleState	472
Table 514 – SubmoduleState.AddInfo	473
Table 515 – SubmoduleState.Advice	473
Table 516 – SubmoduleState.MaintenanceRequired	473

Table 517 – SubmoduleState.MaintenanceDemanded	473
Table 518 – SubmoduleState.Fault	474
Table 519 – SubmoduleState.ARInfo	474
Table 520 – SubmoduleState.IdentInfo	474
Table 521 – SubmoduleState.FormatIndicator.....	475
Table 522 – SubmoduleProperties.Type.....	475
Table 523 – SubmoduleProperties.SharedInput	475
Table 524 – SubmoduleProperties.ReduceInputSubmoduleDataLength	476
Table 525 – SubmoduleProperties.ReduceOutputSubmoduleDataLength.....	476
Table 526 – SubmoduleProperties.DiscardIOXS	476
Table 527 – SubstitutionMode.....	477
Table 528 – SubstituteActiveFlag	477
Table 529 – InitiatorUDPRTPort.....	478
Table 530 – ResponderUDPRTPort.....	478
Table 531 – InitiatorRPCServerPort	478
Table 532 – ResponderRPCServerPort	479
Table 533 – MaxAlarmDataLength	479
Table 534 – APStructureIdentifier with API==0	480
Table 535 – APStructureIdentifier with API ≠ 0.....	480
Table 536 – ExtendedIdentificationVersionHigh	480
Table 537 – ExtendedIdentificationVersionLow	480
Table 538 – Values of ErrorCode for negative responses.....	481
Table 539 – Values of ErrorDecode	482
Table 540 – Coding of ErrorCode1 with ErrorDecode PNORW	482
Table 541 – Coding of ErrorCode2 with ErrorDecode PNORW	483
Table 542 – Coding of ErrorCode1 with ErrorDecode:= PNIO	484
Table 543 – Values of ErrorCode2 for ErrorDecode:= PNIO and ErrorCode1 (part 1).....	487
Table 544 – Values of ErrorCode2 for ErrorDecode:= PNIO and ErrorCode1 (part 2 – alarm acknowledge).....	490
Table 545 – Values of ErrorCode2 for ErrorDecode:= PNIO and ErrorCode1 (part 3 – machines).....	491
Table 546 – Values of ErrorCode2 for ErrorDecode:= PNIO and ErrorCode1 (part 4 – IO controller)	492
Table 547 – Values of ErrorCode2 for ErrorDecode:= PNIO and ErrorCode1 (part 5 – IO device).....	494
Table 548 – Values of ErrorCode2 for ErrorDecode:= PNIO and ErrorCode1 (part 6 – abort reasons)	495
Table 549 – Values of ErrorCode2 for ErrorDecode:= PNIO and ErrorCode1 (part 7 – Reserved).....	498
Table 550 – Coding of ErrorCode1 for ErrorDecode with the value ManufacturerSpecific	498
Table 551 – Coding of ErrorCode2 for ErrorDecode with the value ManufacturerSpecific	498
Table 552 – Visible characters.....	498
Table 553 – FactoryReset / ResetToFactory behavior (legacy from IEC 61158-x-3)	499

Table 554 – FactoryReset / ResetToFactory behavior (default without IEC 61158-x-3 history)	499
Table 555 – FactoryReset / ResetToFactory behavior if used in conjunction with functional safety submodules	499
Table 556 – IM_Hardware_Revision	499
Table 557 – IM_SWRevision_Functional_Enhancement	500
Table 558 – IM_SWRevision_Bug_Fix	500
Table 559 – IM_SWRevision_Internal_Change	500
Table 560 – IM_Revision_Counter	500
Table 561 – IM_Profile_ID	501
Table 562 – IM_Profile_Specific_Type in conjunction with IM_Profile_ID == 0x0000	501
Table 563 – IM_Profile_Specific_Type in conjunction with IM_Profile_ID range 0x0001 – 0xF6FF	501
Table 564 – IM_Version_Major	502
Table 565 – IM_Version_Minor	502
Table 566 – IM_Supported.I&M1	502
Table 567 – IM_Date with time	504
Table 568 – IM_Date without time	504
Table 569 – IM_Annotation	504
Table 570 – IM_OrderID	505
Table 571 – IM_UniquelIdentifier	505
Table 572 – UserStructureIdentifier	506
Table 573 – ChannelErrorType – range 1	508
Table 574 – ChannelErrorType – range 2	509
Table 575 – ChannelErrorType – range 3	509
Table 576 – ChannelErrorType – range 4	510
Table 577 – ChannelNumber	510
Table 578 – ChannelProperties.Type	511
Table 579 – ChannelProperties.Accumulative	511
Table 580 – ChannelProperties.Maintenance	512
Table 581 – Valid combinations within ChannelProperties	512
Table 582 – Valid combinations for Alarmnotification and RecordDataRead(DiagnosisData)	513
Table 583 – ChannelProperties.Specifier	514
Table 584 – ChannelProperties.Direction	514
Table 585 – ExtChannelErrorType	514
Table 586 – Allowed combinations of ChannelErrorType, ExtChannelErrorType, and ExtChannelAddValue	515
Table 587 – ExtChannelErrorType for ChannelErrorType 0 – 0xFF	515
Table 588 – Additional ExtChannelErrorType for ChannelErrorType 0x0F and 0x10	515
Table 589 – ExtChannelErrorType for ChannelErrorType 0x0100 – 0x7FFF	515
Table 590 – ExtChannelErrorType for ChannelErrorType “Data transmission impossible”	516
Table 591 – ExtChannelErrorType for ChannelErrorType “Remote mismatch”	516

Table 592 – ExtChannelErrorType for ChannelErrorType “Media redundancy mismatch – Ring”	517
Table 593 – ExtChannelErrorType for ChannelErrorType “Media redundancy mismatch – Interconnection”	517
Table 594 – ExtChannelErrorType for ChannelErrorType “Sync mismatch” and for ChannelErrorType “Time mismatch”	518
Table 595 – ExtChannelErrorType for ChannelErrorType “Isochronous mode mismatch”	518
Table 596 – ExtChannelErrorType for ChannelErrorType “Multicast CR mismatch”	518
Table 597 – ExtChannelErrorType for ChannelErrorType “Fiber optic mismatch”	519
Table 598 – ExtChannelErrorType for ChannelErrorType “Network component function mismatch”	519
Table 599 – ExtChannelErrorType for ChannelErrorType “Dynamic Frame Packing function mismatch”	519
Table 600 – ExtChannelErrorType for ChannelErrorType “Media redundancy with planned duplication mismatch”	520
Table 601 – ExtChannelErrorType for ChannelErrorType “Multiple interface mismatch”	520
Table 602 – Values for ExtChannelAddValue	521
Table 603 – Values for “Accumulative Info”	521
Table 604 – Values for ExtChannelErrorType “Parameter fault detail”	522
Table 605 – Values for ExtChannelAddValue.Index	522
Table 606 – Values for ExtChannelAddValue.Offset	522
Table 607 – Values for ExtChannelErrorType “Consistency fault detail”	522
Table 608 – Values for ExtChannelAddValue.Index	523
Table 609 – Values for “Fiber optic mismatch” – “Power Budget”	523
Table 610 – Values for “Network component function mismatch” – “Frame dropped”	523
Table 611 – Values for “Remote mismatch” – “Peer CableDelay mismatch”	524
Table 612 – Values for “Multiple interface mismatch” – “Conflicting MultipleInterfaceMode.NameOfDevice mode”	524
Table 613 – Values for “Multiple interface mismatch” – “Inactive StandardGateway”	524
Table 614 – Values for QualifiedChannelQualifier	525
Table 615 – Values for MaintenanceStatus	525
Table 616 – URRecordIndex	527
Table 617 – URRecordLength	527
Table 618 – iPar_Req_Header	527
Table 619 – Max_Segm_Size	527
Table 620 – Transfer_Index	528
Table 621 – Total_iPar_Size	528
Table 622 – MultipleInterfaceMode.NameOfDevice	528
Table 623 – NumberOfPeers in conjunction with PDPortDataCheck	529
Table 624 – NumberOfPeers in conjunction with PDPortDataReal or PDPortDataRealExtended	529
Table 625 – LineDelay.Value with LineDelay.FormatIndicator == 0	530
Table 626 – LineDelay.Value with LineDelay.FormatIndicator == 1	530
Table 627 – LineDelay.FormatIndicator	531
Table 628 – RxPort	531
Table 629 – NumberOfTxPortGroups	531

Table 630 – TxPortEntry	532
Table 631 – FrameDetails.SyncFrame in conjunction with FrameDataProperties.ForwardingMode=="Absolute mode"	533
Table 632 – FrameDetails.SyncFrame in conjunction with FrameDataProperties.ForwardingMode=="Relative mode"	533
Table 633 – FrameDetails.MeaningFrameSendOffset	534
Table 634 – FrameDetails.MediaRedundancyWatchDog	534
Table 635 – FrameDataProperties.ForwardingMode	534
Table 636 – FrameDataProperties.FastForwardingMulticastMACAdd	534
Table 637 – FrameDataProperties.FragmentationMode	535
Table 638 – MAUType	535
Table 639 – MAUType with MAUTypeExtension	539
Table 640 – Valid combinations between MAUType and LinkState	539
Table 641 – MAUTypeExtensions and its corresponding MAUTypes	540
Table 642 – CheckSyncMode.CableDelay	540
Table 643 – CheckSyncMode.SyncMaster	540
Table 644 – MAUTypeMode.Check	541
Table 645 – DomainBoundaryIngress	541
Table 646 – DomainBoundaryEgress	541
Table 647 – DomainBoundaryAnnounce	542
Table 648 – MulticastBoundary	542
Table 649 – PeerToPeerBoundary	543
Table 650 – DCPBoundary	543
Table 651 – PreambleLength.Length	544
Table 652 – LinkState.Link	544
Table 653 – LinkState.Port	545
Table 654 – MediaType	545
Table 655 – MaxBridgeDelay	545
Table 656 – NumberOfPorts	546
Table 657 – MaxPortTxDelay	546
Table 658 – MaxPortRxDelay	546
Table 659 – MaxLineRxDelay	546
Table 660 – YellowTime	547
Table 661 – StartOfRedFrameID in conjunction with ARProperties.StartupMode:= Legacy	549
Table 662 – StartOfRedFrameID in conjunction with ARProperties.StartupMode:= Advanced	550
Table 663 – EndOfRedFrameID	550
Table 664 – Dependencies of StartOfRedFrameID and EndOfRedFrameID	550
Table 665 – NumberOfAssignments	550
Table 666 – NumberOfPhases	551
Table 667 – AssignedValueForReservedBegin	551
Table 668 – AssignedValueForOrangeBegin	552
Table 669 – AssignedValueForReservedEnd	552

Table 670 – Values of RedOrangePeriodBegin	552
Table 671 – Dependencies of RedOrangePeriodBegin, OrangePeriodBegin and GreenPeriodBegin	553
Table 672 – Values of OrangePeriodBegin with ARProperties.StartupMode == Legacy	553
Table 673 – Values of OrangePeriodBegin with ARProperties.StartupMode == Advanced	553
Table 674 – Values of GreenPeriodBegin	553
Table 675 – EtherType	554
Table 676 – SyncProperties.Role	554
Table 677 – SyncProperties.SyncID	554
Table 678 – ReservedIntervalBegin with ARProperties.StartupMode == Legacy	555
Table 679 – ReservedIntervalBegin with ARProperties.StartupMode == Advanced	555
Table 680 – ReservedIntervalEnd with ARProperties.StartupMode == Legacy	555
Table 681 – ReservedIntervalEnd with ARProperties.StartupMode == Advanced	555
Table 682 – Dependencies of ReservedIntervalBegin and ReservedIntervalEnd	555
Table 683 – SyncSendFactor	556
Table 684 – PTCPTimeoutFactor	557
Table 685 – PTCPTakeoverTimeoutFactor	557
Table 686 – PTCPMasterStartupTime	558
Table 687 – PLLWindow	558
Table 688 – TimeIObase	560
Table 689 – TimeDataCycle	560
Table 690 – TimeIOInput	560
Table 691 – TimeIOOutput	561
Table 692 – TimeIOInputValid	561
Table 693 – TimeIOOutputValid	561
Table 694 – ControllerApplicationCycleFactor	561
Table 695 – TimePLLWindow	562
Table 696 – TimeMasterPriority1	563
Table 697 – TimeMasterPriority2	563
Table 698 – MRP_Version	564
Table 699 – MRP_RingState	564
Table 700 – MRP_DomainUUID	564
Table 701 – MRP_LengthDomainName	565
Table 702 – MRP_DomainName	565
Table 703 – MRP_Role	565
Table 704 – MRP_Version	565
Table 705 – MRP_Prio	566
Table 706 – MRP_TOPchgT	566
Table 707 – MRP_TOPNRmax	567
Table 708 – MRP_TSTshortT	567
Table 709 – MRP_TSTdefaultT	567
Table 710 – MRP_TSTNRmax	568
Table 711 – MRP_LNKdownT	568

Table 712 – MRP_LNKupT	568
Table 713 – MRP_LNKNRmax	569
Table 714 – MRP_Check.MediaRedundancyManager	569
Table 715 – MRP_Check.MRP_DomainUUID	569
Table 716 – MRP_NumberOfEntries	570
Table 717 – MRP_Instance	570
Table 718 – MRPIC_LengthDomainName	570
Table 719 – MRPIC_DomainName	570
Table 720 – MRPIC_State	571
Table 721 – MRPIC_Role	571
Table 722 – MRPIC_DomainID	571
Table 723 – MRPIC_TOPchgT	572
Table 724 – MRPIC_TOPNRmax	572
Table 725 – MRPIC_LinkStatusChangeT	573
Table 726 – MRPIC_LinkStatusNRmax	573
Table 727 – MRPIC_LNKdownT	573
Table 728 – MRPIC_LNKupT	574
Table 729 – MRPIC_LNKNRmax	574
Table 730 – MRPIC_StartDelay	575
Table 731 – MRPIC_Check.MIM	575
Table 732 – MRPIC_Check.MRPIC_DomainID	575
Table 733 – VendorBlockType	576
Table 734 – FiberOpticType	576
Table 735 – FiberOpticCableType	576
Table 736 – FiberOpticPowerBudgetType.Value	577
Table 737 – FiberOpticPowerBudgetType.CheckEnable	577
Table 738 – NCDropBudgetType.Value	577
Table 739 – NCDropBudgetType.CheckEnable	578
Table 740 – CounterStatus.ifInOctets	579
Table 741 – CounterStatus.ifOutOctets	579
Table 742 – CounterStatus.ifInDiscards	579
Table 743 – CounterStatus.ifOutDiscards	579
Table 744 – CounterStatus.ifInErrors	579
Table 745 – CounterStatus.ifOutErrors	580
Table 746 – CounterStatus.Reserved	580
Table 747 – FSHelloMode.Mode	581
Table 748 – FSHelloInterval	581
Table 749 – FSHelloRetry	582
Table 750 – FSHelloDelay	582
Table 751 – FSPParameterMode.Mode	582
Table 752 – FSPParameterUUID	583
Table 753 – NumberOfSubframeBlocks	583
Table 754 – SFIOCRProperties.DistributedWatchDogFactor	583

Table 755 – SFIOCRProperties.RestartFactorForDistributedWD	584
Table 756 – SFIOCRProperties.DFPMODE	584
Table 757 – SFIOCRProperties.DFPDirection	585
Table 758 – SFIOCRProperties.DFPRedundantPathLayout.....	585
Table 759 – SFIOCRProperties.SFCRC16	585
Table 760 – SubframeData.Position	586
Table 761 – SubframeData.DataLength	586
Table 762 – Event function table.....	587
Table 763 – SubframeOffset	587
Table 764 – Event function table.....	588
Table 765 – SCFEntry.....	589
Table 766 – ACCommunicationProperties.DFP	590
Table 767 – ACCommunicationProperties.RTC3	590
Table 768 – ACCommunicationProperties.RTCUDP	590
Table 769 – ACMinDeviceInterval	591
Table 770 – FromOffsetData	591
Table 771 – NextOffsetData.....	591
Table 772 – TotalSize	591
Table 773 – RedundancyInfo.EndPoint1	592
Table 774 – RedundancyInfo.EndPoint2	592
Table 775 – Valid combination of RedundancyInfo.EndPoint1 and RedundancyInfo.EndPoint2.....	592
Table 776 – SRProperties.InputValidOnBackupAR with SRProperties.Mode == 0	593
Table 777 – SRProperties.InputValidOnBackupAR with SRProperties.Mode == 1	594
Table 778 – SRProperties.Reserved_1	594
Table 779 – SRProperties.Mode	594
Table 780 – RedundancyDataHoldFactor	594
Table 781 – NumberOfEntries.....	595
Table 782 – PE_OperationalMode	595
Table 783 – AM_Location.Structure	596
Table 784 – AM_Location.Levelx	596
Table 785 – AM_Location.Reserved1.....	597
Table 786 – AM_Location.BeginSubslotNumber.....	597
Table 787 – AM_Location.EndSubslotNumber	597
Table 788 – AM_Location.Reserved2.....	597
Table 789 – AM_Location.Reserved3.....	598
Table 790 – AM_Location.Reserved4.....	598
Table 791 – AM_DeviceIdentification.DeviceSubID	598
Table 792 – AM_DeviceIdentification.DeviceSubID for AM_DeviceIdentification.Organization:= 0x0000	599
Table 793 – AM_DeviceIdentification.DeviceID	599
Table 794 – AM_DeviceIdentification.VendorID.....	599
Table 795 – AM_DeviceIdentification.Organization	599
Table 796 – RS_Properties.AlarmTransport	600

Table 797 – RS_BlockType used for events	600
Table 798 – RS_BlockType used for adjust	601
Table 799 – RS_BlockLength in conjunction with RS_EventBlock	601
Table 800 – RS_BlockLength in conjunction with other blocks	602
Table 801 – RS_Specifier.SequenceNumber	602
Table 802 – RS_Specifier.Specifier	602
Table 803 – RS_MinusError	603
Table 804 – RS_PlusError	603
Table 805 – RS_ExtensionBlockType	603
Table 806 – RS_ExtensionBlockLength	603
Table 807 – RS_MaxScanDelay	604
Table 808 – RS_AdjustSpecifier.Incident	604
Table 809 – RS_ReasonCode.Reason	604
Table 810 – RS_ReasonCode.Detail	605
Table 811 – RS_DigitalInputCurrentValue.Value	605
Table 812 – RS_DomainIdentification	605
Table 813 – RS_MasterIdentification	605
Table 814 – ActualLocalTimeStamp	606
Table 815 – LocalTimeStamp	606
Table 816 – NumberOfLogEntries	606
Table 817 – EntryDetail	606
Table 818 – Time_TimeStamp	607
Table 819 – Allowed combinations of PRAL_Reason, PRAL_ExtReason, and PRAL_ReasonAddValue	607
Table 820 – PRAL_ChannelProperties.Reserved_1	607
Table 821 – PRAL_ChannelProperties.Accumulative	608
Table 822 – PRAL_ChannelProperties.Reserved_2	608
Table 823 – PRAL_ChannelProperties.Direction	608
Table 824 – Values for PRAL_Reason	608
Table 825 – Values for PRAL_ExtReason	610
Table 826 – Usage of PRAL_ReasonAddValue	610
Table 827 – Values for PRAL_ReasonAddValue[0..3]	610
Table 828 – Values for PRAL_ReasonAddValue[0] to [127]	610
Table 829 – ArgsLength check	611
Table 830 – ARBlockReq – request check	612
Table 831 – IOCRBlockReq – request check	613
Table 832 – AlarmCRBlockReq – request check	617
Table 833 – ExpectedSubmoduleBlockReq – request check	617
Table 834 – PrmServerBlock – request check	619
Table 835 – MCRBlockReq – request check	619
Table 836 – ARRPCBlockReq – request check	620
Table 837 – IRInfoBlock – request check	620
Table 838 – SRInfoBlock – request check	621

Table 839 – RSInfoBlock – request check	621
Table 840 – ArgsLength check	622
Table 841 – ARBlockRes – response check	622
Table 842 – IOCRBlockRes – response check	623
Table 843 – AlarmCRBlockRes – response check	624
Table 844 – ModuleDiffBlock – response check	624
Table 845 – ARServerBlockRes – response check	625
Table 846 – ArgsLength check	626
Table 847 – ControlBlockConnect(PrmEnd) – request check	626
Table 848 – ControlBlockPlug(PrmEnd) – request check	627
Table 849 – ControlBlockConnect(PrmBegin) – request check	627
Table 850 – SubmoduleListBlock – request check	628
Table 851 – ArgsLength check	628
Table 852 – ControlBlockConnect – response check	629
Table 853 – ControlBlockPlug – response check	629
Table 854 – ControlBlockConnect(PrmBegin) – response check	630
Table 855 – ArgsLength check	631
Table 856 – ControlBlockConnect(AppIRdy) – request check	631
Table 857 – ControlBlockPlug(AppIRdy) – request check	632
Table 858 – ArgsLength check	632
Table 859 – ControlBlockConnect – response check	633
Table 860 – ControlBlockPlug – response check	633
Table 861 – ArgsLength check	634
Table 862 – ReleaseBlock – request check	635
Table 863 – ArgsLength check	635
Table 864 – ReleaseBlock – response check	636
Table 865 – ArgsLength check	636
Table 866 – IODWriteReqHeader – request check	637
Table 867 – ArgsLength check	637
Table 868 – IODWriteResHeader – response check	638
Table 869 – ArgsLength check	639
Table 870 – ArgsLength check	640
Table 871 – ArgsLength check	640
Table 872 – IODReadReqHeader – request check	641
Table 873 – RecordDataReadQuery – request check	642
Table 874 – ArgsLength check	642
Table 875 – IODReadResHeader – response check	642
Table 876 – Primitives issued by AP-Context (FAL user) to FSPMDEV	645
Table 877 – Primitives issued by FSPMDEV to AP-Context (FAL user)	648
Table 878 – Functions, Macros, Timers and Variables used by the AP-Context (FAL user) to FSPMDEV	651
Table 879 – Functions, Macros, Timers and Variables used by the FSPMDEV to AP-Context (FAL user)	652
Table 880 – Primitives issued by AP-Context (FAL user) to FSPMCTL	655

Table 881 – Primitives issued by FSPMCTL to AP-Context (FAL user).....	657
Table 882 – Functions, Macros, Timers and Variables used by AP-Context (FAL user) to FSPMCTL	661
Table 883 – Functions, Macros, Timers and Variables used by FSPMCTL to AP-Context (FAL user)	662
Table 884 – Remote primitives issued or received by ALPMI	665
Table 885 – Local primitives issued or received by ALPMI	666
Table 886 – ALPMI state table	667
Table 887 – Functions, Macros, Timers and Variables used by ALPMI	668
Table 888 – Remote primitives issued or received by ALPMR	669
Table 889 – Local primitives issued or received by ALPMR	670
Table 890 – ALPMR state table	671
Table 891 – Functions, Macros, Timers and Variables used by ALPMR	673
Table 892 – Remote primitives issued or received by CMDEV	677
Table 893 – Local primitives issued or received by CMDEV	679
Table 894 – CMDEV state table	682
Table 895 – Functions, Macros, Timers and Variables used by CMDEV	684
Table 896 – Remote primitives issued or received by CMDEV_DA	686
Table 897 – Local primitives issued or received by CMDEV_DA	686
Table 898 – CMDEV_DA state table	688
Table 899 – Functions, Macros, Timers and Variables used by CMDEV(DA)	688
Table 900 – Remote primitives issued or received by CMSU	689
Table 901 – Local primitives issued or received by CMSU	689
Table 902 – CMSU state table	692
Table 903 – Functions, Macros, Timers and Variables used by the CMSU	695
Table 904 – Remote primitives issued or received by CMIO	695
Table 905 – Local primitives issued or received by CMIO	695
Table 906 – CMIO state table	697
Table 907 – Functions used by the CMIO	698
Table 908 – Remote primitives issued or received by CMRS	698
Table 909 – Local primitives issued or received by CMRS	699
Table 910 – CMRS state table	700
Table 911 – Functions, Macros, Timers and Variables used by the CMRS	700
Table 912 – Remote primitives issued or received by CMWRR	701
Table 913 – Local primitives issued or received by CMWRR	701
Table 914 – CMWRR state table	703
Table 915 – Functions, Macros, Timers and Variables used by CMWRR	705
Table 916 – Remote primitives issued or received by CMRDR	706
Table 917 – Local primitives issued or received by CMRDR	706
Table 918 – CMRDR state table	707
Table 919 – Functions, Macros, Timers and Variables used by CMRDR	708
Table 920 – Remote primitives issued or received by CMSM	708
Table 921 – Local primitives issued or received by CMSM	709
Table 922 – CMSM state table	710

Table 923 – Functions, Macros, Timers and Variables used by the CMSM	711
Table 924 – Remote primitives received by CMPBE	712
Table 925 – Local primitives issued or received by CMPBE	712
Table 926 – CMPBE state table	714
Table 927 – Functions, Macros, Timers and Variables used by the CMPBE	716
Table 928 – Remote primitives issued or received by CMDMC	716
Table 929 – Local primitives issued or received by CMDMC	717
Table 930 – CMDMC state table	719
Table 931 – Functions, Macros, Timers and Variables used by the CMDMC	721
Table 932 – Remote primitives issued or received by CMINA	722
Table 933 – Local primitives issued or received by CMINA	722
Table 934 – CMINA state table	724
Table 935 – Functions, Macros, Timers and Variables used by the CMINA	730
Table 936 – Return values of CheckAPDU	731
Table 937 – Remote primitives issued or received by CMRPC	732
Table 938 – Local primitives issued or received by CMRPC	734
Table 939 – CMRPC state table	735
Table 940 – Functions, Macros, Timers and Variables used by the CMRPC	738
Table 941 – Return values of CheckRPC	740
Table 942 – Remote primitives issued or received by CMSRL	741
Table 943 – Local primitives issued or received by CMSRL	742
Table 944 – CMSRL state table	743
Table 945 – Functions, Macros, Timers and Variables used by the CMSRL	746
Table 946 – Combinations of DataStatus for Output buffers	747
Table 947 – Combinations of DataStatus for Input buffers	747
Table 948 – Remote primitives issued or received by CMSRL_AL	753
Table 949 – Local primitives issued or received by CMSRL_AL	753
Table 950 – CMSRL_AL state table	755
Table 951 – Functions, Macros, Timers and Variables used by the CMSRL_AL	756
Table 952 – Remote primitives issued or received by CMCTL	759
Table 953 – Local primitives issued or received by CMCTL	760
Table 954 – CMCTL state table	764
Table 955 – Functions, Macros, Timers and Variables used by the CMCTL	767
Table 956 – Remote primitives issued or received by CTLSM	767
Table 957 – Local primitives issued or received by CTLSM	768
Table 958 – CTLSM state table	769
Table 959 – Functions, Macros, Timers and Variables used by the CTLSM	770
Table 960 – Remote primitives issued or received by CTLIO	770
Table 961 – Local primitives issued or received by CTLIO	771
Table 962 – CTLIO state table	772
Table 963 – Functions, Macros, Timers and Variables used by the CTLIO	773
Table 964 – Remote primitives received by CTRLDI	774
Table 965 – Local primitives issued or received by CTRLDI	774

Table 966 – CTLRDI state table	776
Table 967 – Functions, Macros, Timers and Variables used by CTLRDI	776
Table 968 – Remote Primitives received by CTLRDR	777
Table 969 – Local primitives issued or received by CTLRDR	778
Table 970 – CTLRDR state table	778
Table 971 – Functions, Macros, Timers and Variables used by CTLRDR	779
Table 972 – Remote primitives received by CTLRPC	779
Table 973 – Local primitives issued or received by CTLRPC	782
Table 974 – CTLRPC state table	783
Table 975 – Functions, Macros, Timers and Variables used by the CTLRPC	785
Table 976 – Remote primitives issued or received by CTLSU	785
Table 977 – Local Primitives issued or received by CTLSU	786
Table 978 – CTLSU state table	788
Table 979 – Functions, Macros, Timers and Variables used by the CTLSU	790
Table 980 – Remote primitives issued or received by CTLWRI	791
Table 981 – Local primitives issued or received by CTLWRI	791
Table 982 – CTLWRI state table	793
Table 983 – Functions, Macros, Timers and Variables used by CTLWRI	794
Table 984 – Remote primitives issued or received by CTLWRR	795
Table 985 – Local primitives issued or received by CTLWRR	795
Table 986 – CTLWRR state table	797
Table 987 – Functions, Macros, Timers and Variables used by CTLWRR	797
Table 988 – Remote primitives issued or received by CTLPBE	798
Table 989 – Local primitives issued or received by CTLPBE	799
Table 990 – CTLPBE state table	800
Table 991 – Functions, Macros, Timers and Variables used by CTLPBE	802
Table 992 – Remote primitives issued or received by CTLDINA	803
Table 993 – Local primitives issued or received by CTLDINA	804
Table 994 – CTLDINA state table	806
Table 995 – Functions, Macros, Timers and Variables used by the CTLDINA	809
Table 996 – Remote primitives issued or received by CTLSRL	811
Table 997 – Local primitives issued or received by CTLSRL	811
Table 998 – CTLSRL state table	813
Table 999 – Functions, Macros, Timers and Variables used by the CTLSRL	815
Table A.1 – Examples for the AR establishing	819
Table A.2 – Startup of Alarm transmitter and receiver	819
Table B.1 – Examples for compatible AR establishing	828
Table I.1 – Priority regeneration and queue usage	842
Table M.1 – IEEE 802.3 cross reference	849
Table R.1 – Truth table	864
Table R.2 – “MAC/PHY configuration/status” with AutoNegotiation disabled	864
Table R.3 – “MAC/PHY configuration/status” with AutoNegotiation enabled	864
Table R.4 – Auto-negotiation support within “MAC/PHY configuration/status”	864

Table R.5 – Auto-negotiation settings	865
Table T.1 – List of supported MIBs	867
Table U.1 – Content of archive	868
Table W.1 – Cross reference IEC 62439-2 “MRP MIB objects”	887
Table W.2 – Cross reference IEC 62439-2 “Events, created by state machines”	887
Table W.3 – Cross reference IEC 62439-2 “MRM parameter”	888
Table W.4 – Cross reference IEC 62439-2 “MRC parameter”	888
Table W.5 – Cross reference IEC 62439-2 “MRP MIB objects”	888
Table W.6 – Cross reference IEC 62439-2 “Events, created by state machines”	889
Table W.7 – Cross reference IEC 62439-2 “MIM parameter”	889
Table W.8 – Cross reference IEC 62439-2 “MIC parameter”	889
Table X.1 – Meaning of numbers	891
Table X.2 – Statistic counters – octets	892
Table X.3 – Statistic counters – packets or frames	893
Table X.4 – Statistic counters – errors	893

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019

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 IEC 61784-1 and IEC 61784-2.

International Standard IEC 61158-6-10 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This fourth edition cancels and replaces the third edition published in 2014. This edition constitutes a technical revision.

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

- a) integration of system redundancy basic functionality;
- b) integration of dynamic reconfiguration basic functionality;
- c) integration of reporting system basic functionality;
- d) integration of asset management basic functionality; e) integration of media redundancy ring interconnection basic functionality.

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

FDIS	Report on voting
65C/948/FDIS	65C/956/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.

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

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

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

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

A bilingual version of this publication may be issued at a later date.

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 may work together in any combination.

NOTE Attention is drawn to the fact that use of the associated protocol type(s) is restricted by its (their) intellectual-property-right holder(s). In all cases, the commitment to limited release of intellectual-property-rights made by the holder(s) of those rights permits a particular data-link layer protocol type to be used with physical layer and application layer protocols in type combinations as specified explicitly in the IEC 61784 series. Use of the protocol type(s) in other combinations may require permission of their respective intellectual-property-right holder(s).

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 concerning Type 10 elements and possibly other types given in this document as follows:

The following patent rights for Type 10 have been announced by [SI]:

Publication	Title
WO 02/043336	System and method for parallel transfer of real-time critical and non-real-time critical data via switchable data networks, particularly Ethernet
WO 02/076033	Synchronous clocked communication system with decentralized input/output modules and methods for integrating decentralized input/output modules in such a system
WO 03/028258	Method for synchronizing nodes of a communication system
WO 03/028259	Communications system and method for synchronizing a communications cycle
WO 04/030284	Method for permanent redundant transmission of data telegrams in communication systems
EP 1558002	Method for assigning an IP address to a device
EP 1318630	Matrices for controlling the device specific data transfer rates on a field bus

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

The holder of these patent rights has assured the IEC that he/she is willing to negotiate licences either free of charge or 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:

[SI]: Siemens AG
LC TE IP&IT

Otto-Hahn-Ring 6
D-81739 Munich
Germany

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

ISO (www.iso.org/patents) and IEC (http://www.iec.ch/tctools/patent_decl.htm) maintain on-line data bases of patents relevant to their standards. Users are encouraged to consult the data bases for the most up to date information concerning patents.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019

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 some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

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

- a) the abstract syntax defining the application layer protocol data units conveyed between communicating application entities,
- b) the transfer syntax defining the application layer protocol data units conveyed between communicating application entities,
- c) the application context state machine defining the application service behavior visible between communicating application entities, and
- d) 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:

- a) define the wire-representation of the service primitives defined in IEC 61158-5-10 and
- b) 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.

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

IEC 61158-1:2019, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

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

IEC 61158-5-10:2019, *Industrial communication networks – Fieldbus specifications – 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 62439-2, *Industrial communication networks – High availability automation networks – Part 2: Media Redundancy Protocol (MRP)*

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, *Information technology – Open Systems Interconnection – Presentation service definition*

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

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

ISO/IEC 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 60559:2011, *Information technology – Microprocessor Systems – Floating-Point arithmetic*

ISO 8601, *Data elements and interchange formats – Information interchange – Representation of dates and times*

IEEE Std 802, *IEEE Standard for Local and metropolitan area networks: Overview and Architecture*, available at <http://www.ieee.org> [viewed 2018-09-27]

IEEE Std 802.1AB:2016, *IEEE Standard for Local and metropolitan area networks: Station and Media Access Control Connectivity Discovery*, available at <http://www.ieee.org> [viewed 2018-09-27]

IEEE Std 802.1AC, *IEEE Standard for Local and metropolitan area networks – Media Access Control (MAC) Service definition*, available at <http://www.ieee.org> [viewed 2018-09-27]

IEEE Std 802.1AS, *IEEE standard for Local and metropolitan area networks – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*, available at <http://www.ieee.org> [viewed 2018-09-27]

IEEE Std 802.1Q-2018, *IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks*, available at <http://www.ieee.org> [viewed 2018-09-27]

IEEE Std 802.3, *IEEE Standard for Ethernet*, available at <http://www.ieee.org> [viewed 2018-09-27]

IEEE Std 802.11, *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*, available at <http://www.ieee.org> [viewed 2018-09-27]

IEEE Std 802.15.1, *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 15.1: Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)*, available at <http://www.ieee.org> [viewed 2018-09-27]

IETF RFC 768, *User Datagram Protocol*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 791, *Internet Protocol*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 792, *Internet Control Message Protocol*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 826, *An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 950, *Internet Standard Subnetting Procedure*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 1034, *Domain names – Concepts and Facilities*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 2131, *Dynamic Host Configuration Protocol*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 2132, *DHCP Options and BOOTP Vendor Extensions*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 2236, *Internet Group Management Protocol, Version 2*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 2365, *Administratively Scoped IP Multicast*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 2474, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 2674, *Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 2863, *The Interfaces Group MIB*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 3418, *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 3621, *Power Ethernet MIB*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 4361, *Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 4363, *Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering, and Virtual LAN Extensions*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 4604, *Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast*, available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 4632, *Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*, available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 4836, *Definitions of Managed Objects for IEEE Std 802.3 Medium Attachment Units (MAUs)*, available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 5227, *IPv4 Address Conflict Detection*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 5890, *Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 5905, *Network Time Protocol Version 4: Protocol and Algorithms Specification*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 6151, *Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms*; available at <http://www.ietf.org> [viewed 2018-09-27]

IETF RFC 6890, *Special-Purpose IP Address Registries*; available at <http://www.ietf.org> [viewed 2018-09-27]

The Open Group, *Publication C706, Technical standard DCE1.1: Remote Procedure Call*, available at <http://www.opengroup.org/onlinepubs/9629399/toc.htm> [viewed 2018-09-27]

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 for decentralized periphery

3.2.1

alarm

activation of an event that shows a critical state

3.2.2

alarm ack

acknowledgment of an event that shows a critical state

3.2.3

alarm data object

object(s) which represent 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 layer interoperability

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

3.2.7

application objects

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

3.2.8

application process

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

3.2.9

application process identifier

distinguishes multiple application processes used in a device

3.2.10

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 Definition). 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.

3.2.11

application process object class

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

3.2.12**application relationship**

cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation. This relationship is activated either by the exchange of application-protocol-data-units or as a result of preconfiguration activities

3.2.13**application relationship application service element**

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

3.2.14**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.

3.2.15**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 may also affect the behavior of an object. Attributes are divided into class attributes and instance attributes.

3.2.16**backup**

status of the IO AR, which indicates that it is in the standby state

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

3.2.17**behavior**

indication of how an object responds to particular events

3.2.18**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 real interface of a module or submodule. This reference is used to identify points of failure within diagnosis PDUs.

3.2.19**channel diagnosis**

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

EXAMPLE Line break

3.2.20**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.

3.2.21

class attributes

attribute that is shared by all objects within the same class

3.2.22

class code

unique identifier assigned to each object class

3.2.23

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.

3.2.24

clear

status of the IO controller, which indicates that the control algorithm is currently not running

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 across 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 contains 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 relationship and continuous updating of non-changed IO Data objects

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 for 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 Link Fail

3.2.50

frame

unit of data transfer on a link

3.2.51

identification data object

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

3.2.52

implicit AR endpoint

AR endpoint that is defined locally within a device without use of the create service

3.2.53

inbound

input communication relation from IO device to IO controller

3.2.54

index

address of a record data object within an application process

3.2.55

instance

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

3.2.56

instance attributes

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

3.2.57

instantiated

object that has been created in a device

3.2.58**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 may 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.

3.2.59**IO controller**

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

Note 1 to entry: This is usually a programmable controller or a distributed control system.

3.2.60**IO Data object**

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

3.2.61**IO device**

field device which acts as server for IO operation

3.2.62**IO parameter server**

server for application parameter of IO devices (client)

Note 1 to entry: This is usually a device to backup parameter data and to log online changes of device parameter.

3.2.63**IO subsystem**

subsystem composed of one IO controller and all its associated IO devices

3.2.64**IO supervisor**

engineering device which manages commissioning and diagnosis of an IO system

3.2.65**IO system**

system composed of all its IO subsystems

Note 1 to entry: As an example a Programmable Logic Controller with more than one IO controller (network interface) controls one IO system composed of an IO subsystem for each IO controller.

3.2.66**Isochronous mode**

IO system operating tightly synchronized with a jitter of less than 1 μ s

3.2.67**member**

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

3.2.68**message**

synonym for frame

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

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 have clearly defined interface and behavior

3.2.73

object specific service

service unique to the object class which defines it

3.2.74

operate

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

3.2.75

outbound

output communication relation from an IO controller to an IO device

3.2.76

packet

frame

3.2.77

peer

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

3.2.78

physical device

automation or other network device

3.2.79

point-to-point connection

connection that exists between exactly two application objects

3.2.80

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. In example used for redundancy and dynamic reconfiguration of IO Data.

3.2.81

provider

node or source sending data to one or many consumer

3.2.82**PTCP subdomain**

certain amount of DTEs with synchronized clocks

3.2.83**qualified channel diagnosis**

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

EXAMPLE Overload

3.2.84**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

3.2.85**RedApplication**

an application outside of this standard and supporting redundant or high available application

3.2.86**resource**

processing or information capability

3.2.87**run**

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

3.2.88**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

3.2.89**service**

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

3.2.90**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.

3.2.91**stop**

status of the IO controller which indicates that the control algorithm is currently not running

3.2.92**submodule**

hardware or logical component of a module

3.2.93**subslot**

address of a structural unit within a slot

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

3.2.94

vendor ID

central administrative number used as manufacturer identification

3.3 Abbreviated terms and symbols

3.3.1 Abbreviated terms and symbols for media redundancy

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.

3.3.2 Abbreviated terms and symbols for decentralized periphery

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
ASE	Application Service Element
BLOB	Binary Large Object
BMA	Best-Master-Algorithm Protocol Machine
BMC	Best Master Clock
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	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
CMSM	Context Management Surveillance Protocol Machine Device
CMSRL	Context Management System Redundancy Layer Device
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
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
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 Machines
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
Isb	Least Significant Bit
LT	Length/Type
MAC	Media Access Control
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
NAP	Network Access Point
NCA	Network Computing Architecture
OSF	Open Software Foundation
OSI	Open Systems Interconnect
PDU	Protocol Data Unit
PL	Physical Layer
PPM	Provider protocol machine for cyclic transmitted data
PTCP	Precision Transparent Clock Protocol
QoS	Quality of Service
RED_RELAY	Realtime Class 3 Forwarding Protocol Machine
RNG	Random Number Generator
RPC	Remote Procedure Call
RS	Reporting System
RSM	Best Remote Sync Master Protocol Machine
RSO	Reporting System Observer
RSTP	Rapid Spanning Tree Protocol
RT	Real Time Protocol
RT_CLASS_UDP	Realtime Class UDP (supports Layer 3 transport)
RT_CLASS_1	Realtime Class 1 (Layer 2 transport)
RT_CLASS_2	Realtime Class 2 (Layer 2 transport)
RT_CLASS_3	Realtime Class 3 (Layer 2 transport)
RTA	Real Time Protocol Acyclic
RTC	Real Time Protocol Cyclic
RTC3PSM	Realtime Class 3 Port State Machine
S&F	Store and Forward
SCHEDULER	Scheduler Protocol Machine
SDU	Service Data Unit
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
TLV	Type Length Value (coding rule)
UDP	User Datagram Protocol
UUID	Universal Unique Identifier
VLAN	Virtual Local Area Network

3.3.3 Abbreviated terms and symbols for services

cnf	Confirmation
ind	Indication
req	Request
rsp	Response

3.3.4 Abbreviated terms and symbols for IEEE 802.1Q

DEI	Drop Eligible Indicator
PCP	Priority Code Point
TCI	Tag Control Information
VID	VLAN Identifier

3.3.5 Abbreviated terms and symbols for IETF RFC 2474

DSCP	Differentiated Services Code Point
ECN	Explicit Congestion Notification

3.3.6 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 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 decentralized periphery

3.4.2.1 Abstract syntax conventions

3.4.2.1.1 General

PDUs are described as octets or groups of octets:

- 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.
- If octets or groups of octets are grouped within "{ }" the order is arbitrary.
- If octets or groups of octets are marked with "*" they may appear more than once. If it is used within a "{ }" section they may appear mixed with other octets or group of octets of this section.
- 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 The formal PDU example

APDU = Octet1, OctetGroup1, [Octet2], [Octet3], {[OctGroup2*], 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 characterised 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 Convention for the encoding of reserved bits and octets

The term “reserved” may be 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 tested 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” may also be 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 tested 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 codings 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.

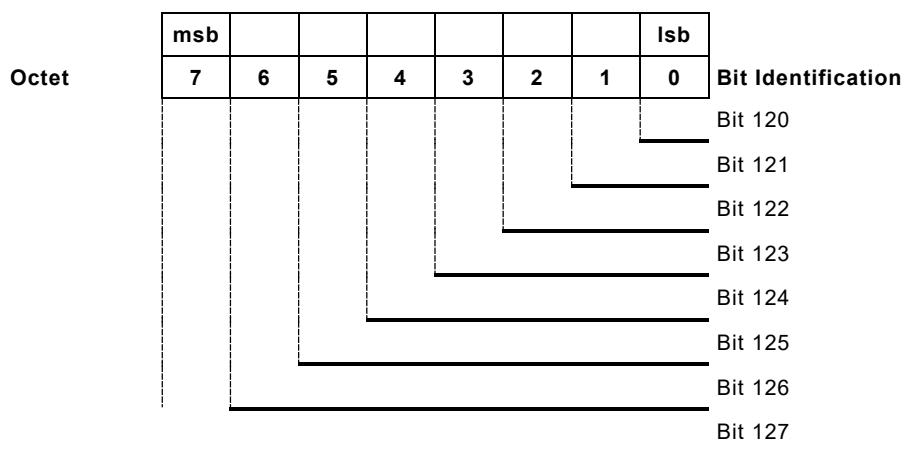


Figure 1 – Common structure of specific fields for octet 1 (high)

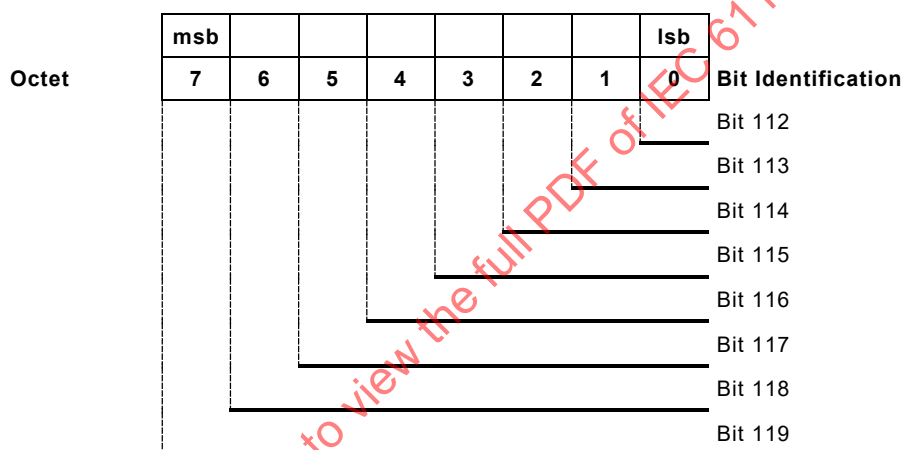


Figure 2 – Common structure of specific fields for octet 2

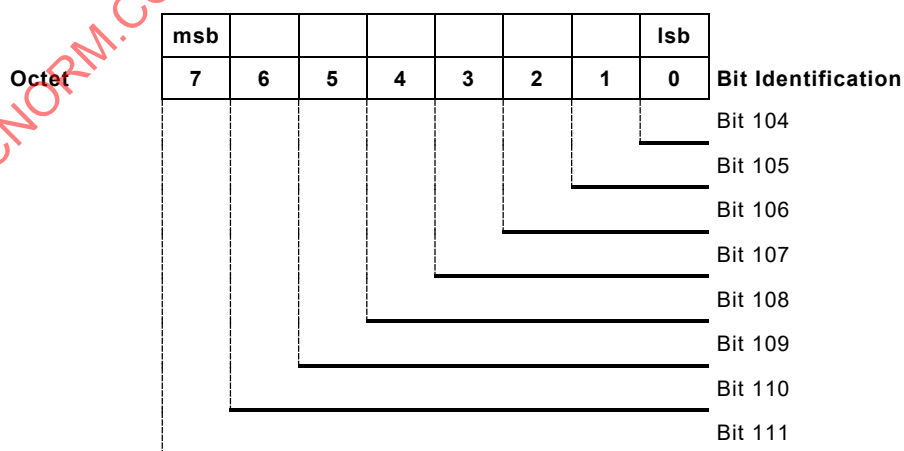


Figure 3 – Common structure of specific fields for octet 3

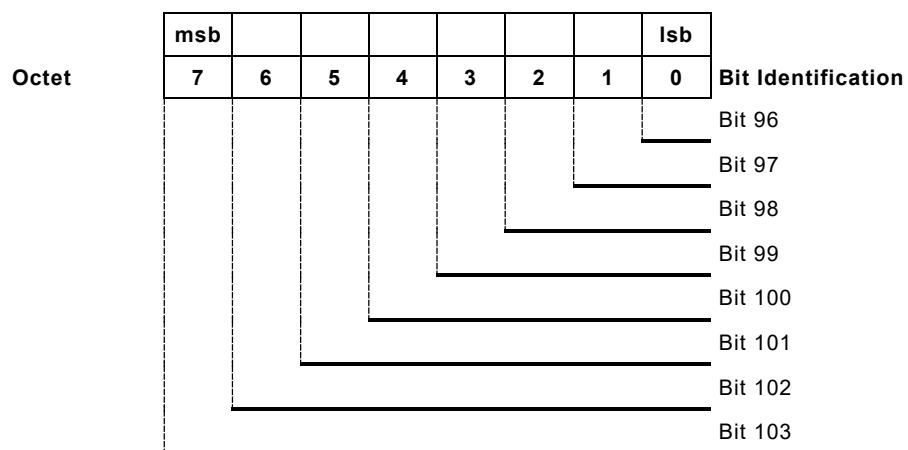


Figure 4 – Common structure of specific fields for octet 4

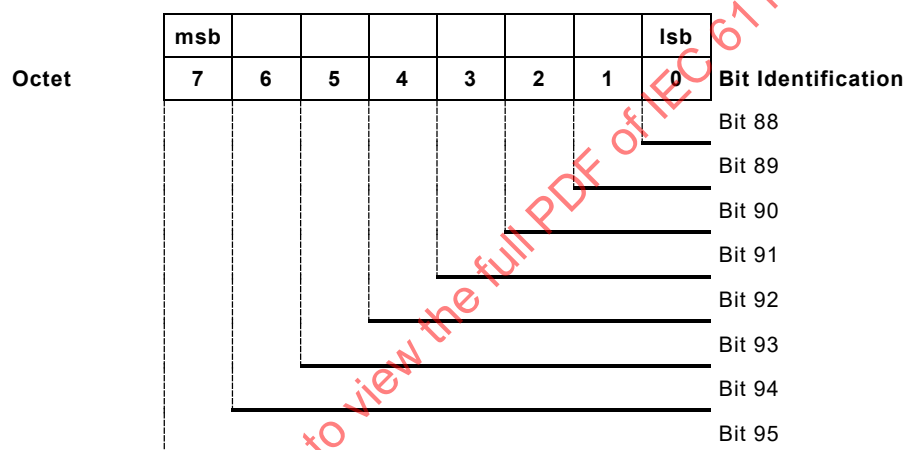


Figure 5 – Common structure of specific fields for octet 5

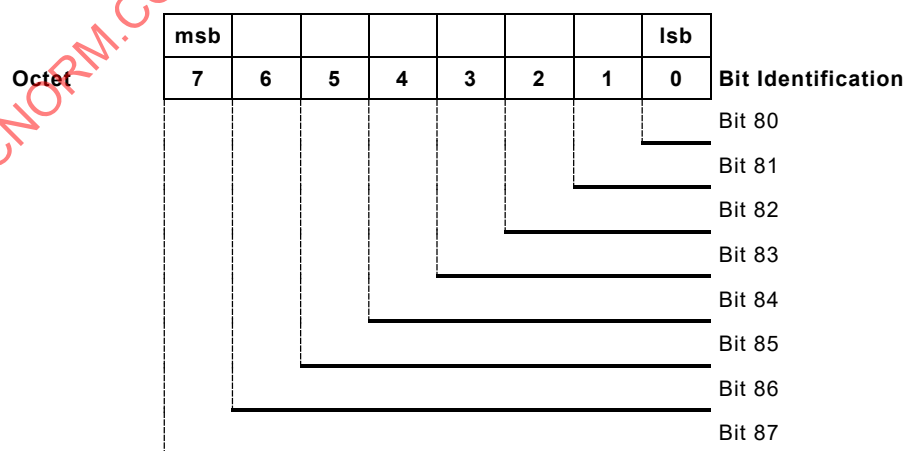


Figure 6 – Common structure of specific fields for octet 6

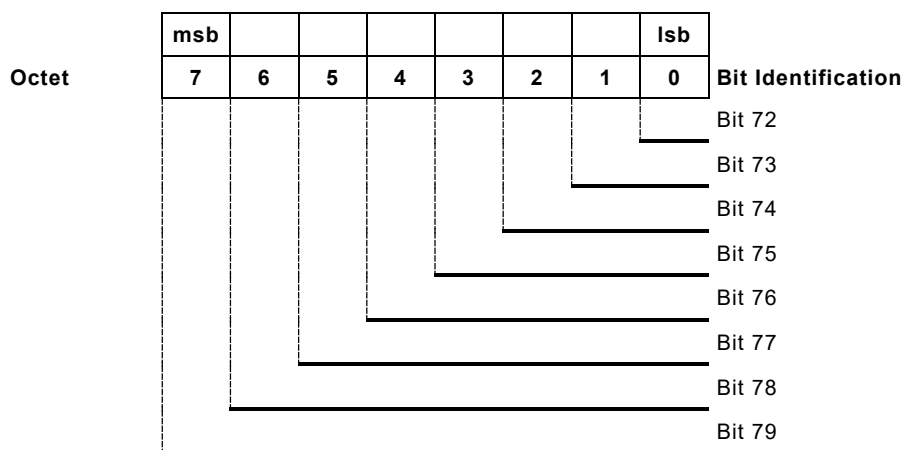


Figure 7 – Common structure of specific fields for octet 7

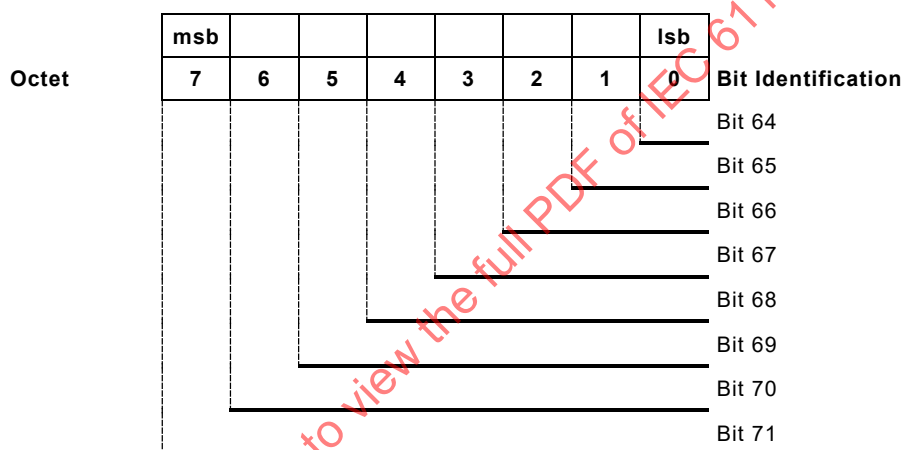


Figure 8 – Common structure of specific fields for octet 8

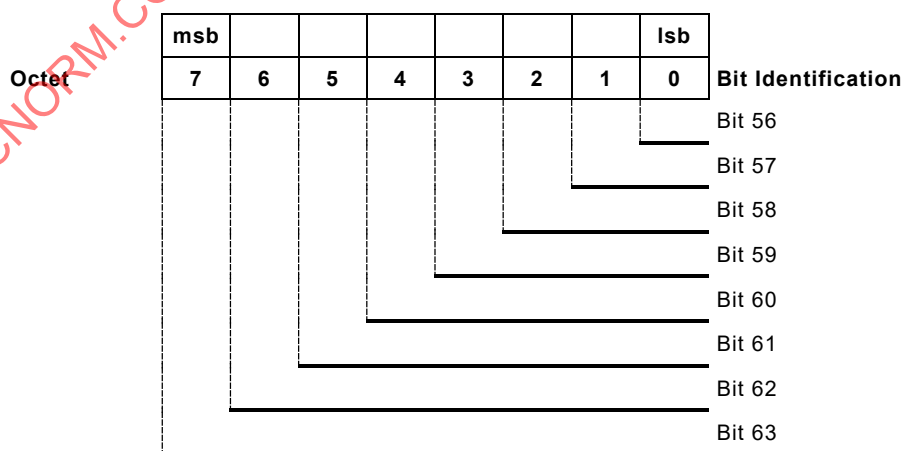


Figure 9 – Common structure of specific fields for octet 9

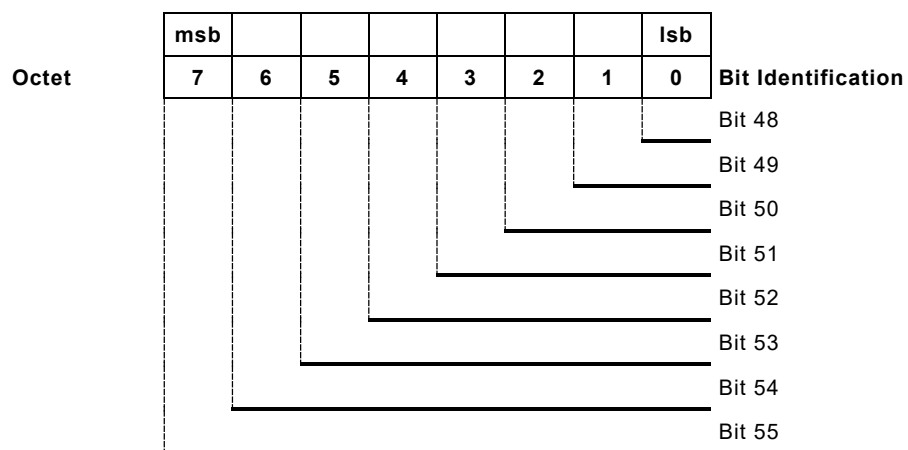


Figure 10 – Common structure of specific fields for octet 10

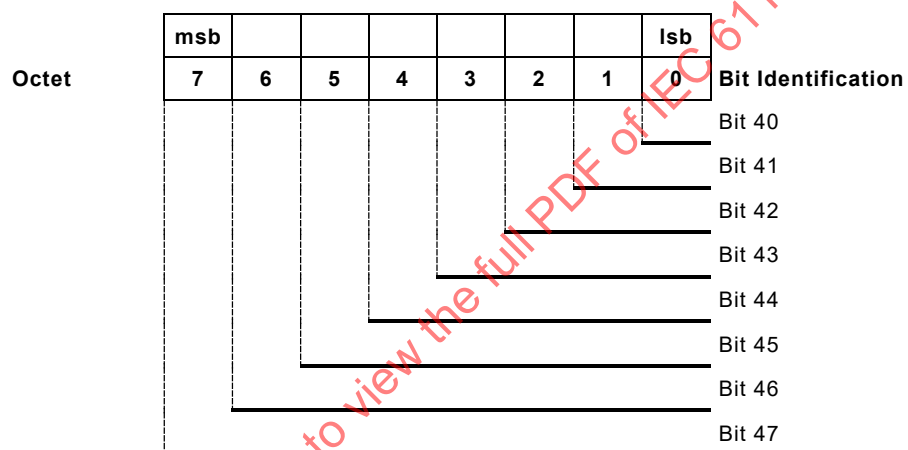


Figure 11 – Common structure of specific fields for octet 11

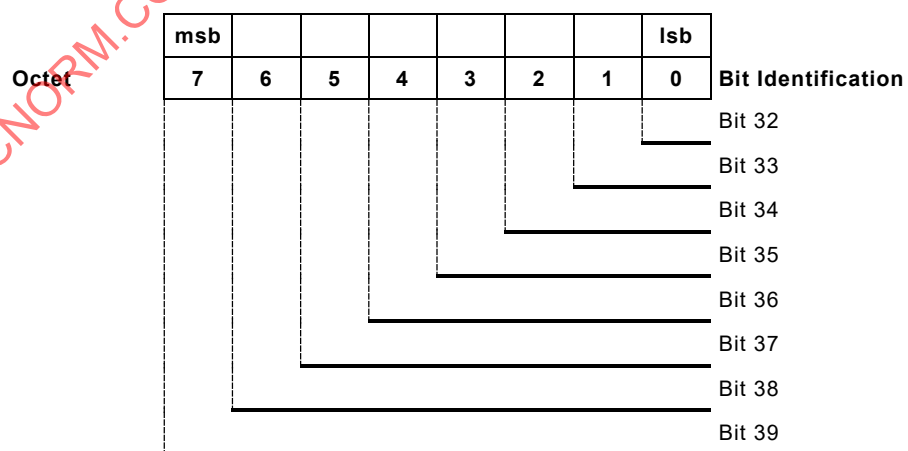


Figure 12 – Common structure of specific fields for octet 12

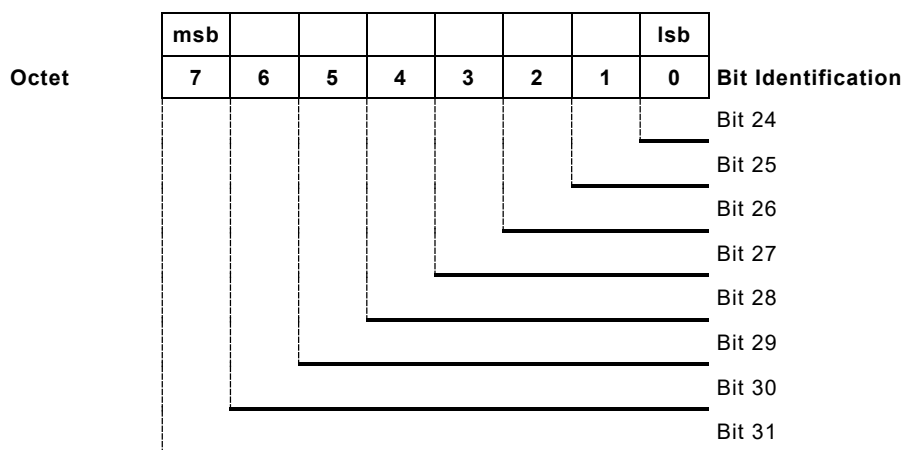


Figure 13 – Common structure of specific fields for octet 13

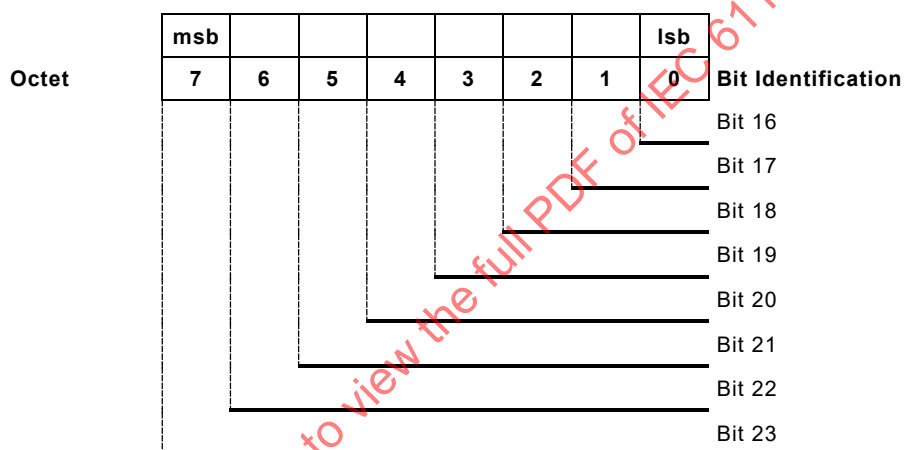


Figure 14 – Common structure of specific fields for octet 14

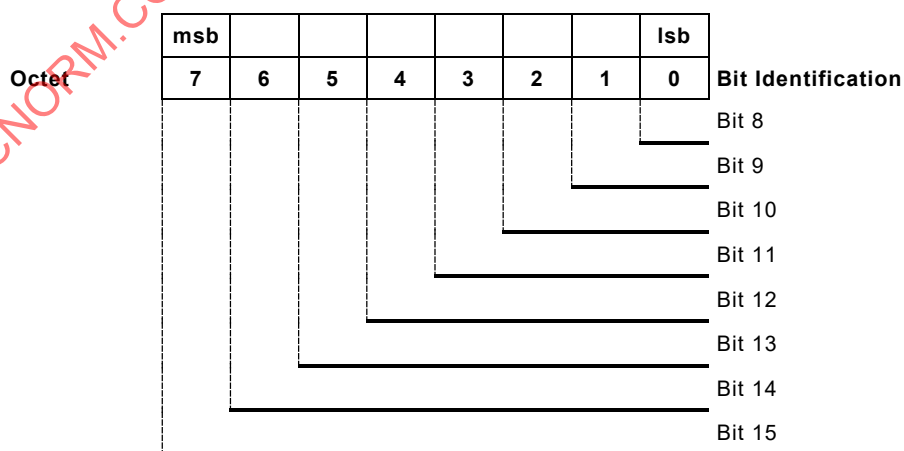


Figure 15 – Common structure of specific fields for octet 15

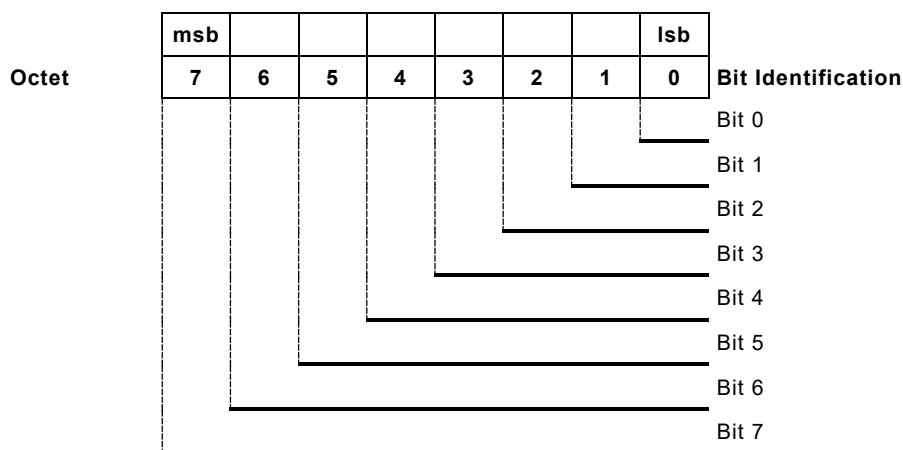


Figure 16 – Common structure of specific fields for octet 16 (low)

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 always 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 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 calling the INIT service. If it is necessary to have different instances of one machine type, different parameters are 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 row contains the name of the transition.
- The second row defines the current state.
- The third row 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 row 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 or condition => action	Next state

Table 7 – Description of state machine elements

Description element	Meaning
Current state	Name of the given states.
Next state	
#	Name or number of the state transition.
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 comes from the primitive shown as an input event.
xxx	A parameter name. Example: Identifier:= reason means value of a 'reason' parameter is assigned to a parameter called 'Identifier.'
"xxx"	Indicates fixed value. Example: Identifier:= "abc" means 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 or less than the item on the right.
>=	A logical condition to indicate an item on the left is equal or greater than the item on the right.
for (Identifier:= start_value to end_value) actions endfor	This construct allows the execution of a sequence of actions in a loop within one transition. The loop is executed for all values from start_value to end_value.
If (condition) actions else actions	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 conventions used for service 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 inside the brackets optionally
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 inside the brackets optionally
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 inside the brackets optionally
(), (+), (-)	replacement of the associated parameter list, described in the primitives definition except local services and functions can deliver parameters inside the brackets if the result is not necessary for further executions, the sign + and – 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 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 may appear 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 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”.

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 place holder 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 place holder 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 802.3, IEEE 802.11 and IEEE 802.15.1.

4.1.1.2 IEEE 802.3

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

Table 10 – IEEE 802.3 DLPDU syntax

DLPDU name	DLPDU structure
DLPDU	Preamble ^a , StartFrameDelimiter, DestinationAddress, SourceAddress, DLSDU ^b , DLPDU_Padding* ^c , FrameCheckSequence
DLSDU	[VLAN] ^d , LT, SAPDU ^ FIDAPDU
SAPDU	UDP-RTC-PDU ^ UDP-RTA-PDU ^ CL-RPC-PDU ^ LLDP-PDU ^e ^ ICMP-PDU
FIDAPDU	FrameID, RTC-PDU ^ RTA-PDU ^ DCP-PDU ^ PTCP-PDU ^e ^ FRAG-PDU
VLAN	LT(=0x8100), TagControllInformation
NOTE According to IEEE 802.3 the DLPDUs have a minimum length of 64 octets (excluded Preamble, Start Frame Delimiter).	
^a IEEE 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 802.3 ^d The VLAN field of DLPDU containing a RTC-PDU should be encoded by the sender, but shall be omitted for RT_CLASS_3. It may be omitted by the conveyance of the frame. The RED_RELAY should discard RT_CLASS_3 frames with VLAN. The decoder shall accept DLPDUs with or without VLAN field. Only one VLAN shall be supported, more VLANs may be supported. ^e The VLAN field shall be omitted except for the PTCP-AnnouncePDU.	

4.1.1.3 IEEE 802.11

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

Table 11 – IEEE 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] ^b , DLSDU, DLPDU_Padding* ^a , FrameCheckSequence
DLPDU_2	FrameControl, DestinationAddress, BSSID, SourceAddress, SequenceControl, [QoSControl] ^b , DLSDU, DLPDU_Padding* ^a , FrameCheckSequence
DLPDU_3	FrameControl, BSSID, SourceAddress, DestinationAddress, SequenceControl, [QoSControl], DLSDU, DLPDU_Padding* ^a , FrameCheckSequence
DLPDU_4	FrameControl, ReceiverAddress, TransmitterAddress, DestinationAddress, SequenceControl, SourceAddress, [QoSControl], DLSDU, DLPDU_Padding* ^a , FrameCheckSequence
DLSDU	DSAP(=0xaa), SSAP(=0xaa), CTRL(=0x03), OUI(=0), [VLAN] ^b , 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
NOTE 1 For definition of FrameControl see IEEE 802.11 NOTE 2 For definition of BSSID see IEEE 802.11 NOTE 3 For definition of SequenceControl see IEEE 802.11 NOTE 4 For definition of QoSControl see IEEE 802.11 NOTE 5 For definition of ReceiverAddress see IEEE 802.11 NOTE 6 For definition of TransmitterAddress see IEEE 802.11 NOTE 7 IEEE 802.11 supports the use of VLANs in conjunction with FrameClassifier Type 2. For definition of FrameClassifier see IEEE 802.11 NOTE 8 For definition of DSAP see IEEE 802.11 NOTE 9 For definition of SSAP see IEEE 802.11 NOTE 10 For definition of CTRL see IEEE 802.11 NOTE 11 For definition of OUI in this context see IEEE 802.11	
^a 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 802.11	
^b The VLAN field of a RTC DLPDU should be encoded by the sender, but omitted for RT_CLASS_3. It may be omitted by the conveyance of the frame. The decoder shall accept DLPDUs with or without VLAN fields.	

4.1.1.4 IEEE 802.15.1

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

Table 12 – IEEE 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	[VLAN] ^b , LT, SAPDU ^c ^ 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), TagControlInformation
NOTE 1 For definition of Access Code see IEEE 802.15.1 NOTE 2 For definition of Packet Header see IEEE 802.15.1 NOTE 3 For definition of Payload Header see IEEE 802.15.1 NOTE 4 For definition of L2CAP Header see IEEE 802.15.1 NOTE 5 For definition of BNEPType see IEEE 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, but omitted for RT_CLASS_3. It may be omitted by the conveyance of the frame, but shall be omitted by the RED_RELAY. 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 $-2^7 \leq I \leq 2^7-1$

Integer16 ::= INTEGER (-32 768..+32 767) -- range $-2^{15} \leq I \leq 2^{15}-1$

Integer32 ::= INTEGER -- range $-2^{31} \leq I \leq 2^{31}-1$

Integer64 ::= INTEGER -- range $-2^{63} \leq I \leq 2^{63}-1$

4.1.2.3 Notation for the Unsigned type

Unsigned8 ::= INTEGER (0..255) -- range $0 \leq I \leq 2^8-1$

Unsigned16 ::= INTEGER (0..65 535) -- range $0 \leq I \leq 2^{16}-1$

Unsigned32 ::= INTEGER -- range $0 \leq I \leq 2^{32}-1$

Unsigned64 ::= INTEGER -- range $0 \leq I \leq 2^{64}-1$

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 802.1AS derived time format

4.1.2.9 Notation for TimeStampDifference type

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

4.1.2.10 Notation for TimeStampDifferenceShort type

TimeStampDifferenceShort::= OCTET STRING SIZE (8) -- IEEE 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 otherwise stated.

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 significand 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								Fractional portion of seconds in $1 / 2^{32}$
	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}	
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.
 NetworkTime₁ is the minuend
 NetworkTime₂ 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								Status
2	2 ¹⁵	2 ¹⁴	2 ¹³	2 ¹²	2 ¹¹	2 ¹⁰	2 ⁹	2 ⁸	
3	Reserved						Time source		
4	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	Seconds
5	2 ⁴⁷	2 ⁴⁶	2 ⁴⁵	2 ⁴⁴	2 ⁴³	2 ⁴²	2 ⁴¹	2 ⁴⁰	
6	2 ³⁹	2 ³⁸	2 ³⁷	2 ³⁶	2 ³⁵	2 ³⁴	2 ³³	2 ³²	
7	2 ³¹	2 ³⁰	2 ²⁹	2 ²⁸	2 ²⁷	2 ²⁶	2 ²⁵	2 ²⁴	
8	2 ²³	2 ²²	2 ²¹	2 ²⁰	2 ¹⁹	2 ¹⁸	2 ¹⁷	2 ¹⁶	
9	2 ¹⁵	2 ¹⁴	2 ¹³	2 ¹²	2 ¹¹	2 ¹⁰	2 ⁹	2 ⁸	
10	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	Nanoseconds
11	2 ³¹	2 ³⁰	2 ²⁹	2 ²⁸	2 ²⁷	2 ²⁶	2 ²⁵	2 ²⁴	
12	2 ²³	2 ²²	2 ²¹	2 ²⁰	2 ¹⁹	2 ¹⁸	2 ¹⁷	2 ¹⁶	
	2 ¹⁵	2 ¹⁴	2 ¹³	2 ¹²	2 ¹¹	2 ¹⁰	2 ⁹	2 ⁸	
	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	Meaning
0	TimeStamp related to global synchronized (global time timescale) time
1	TimeStamp related to local (derived from global time timescale) time
2	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.
 TimeStamp₁ is the minuend
 TimeStamp₂ 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								Nanoseconds
	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}	
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.

TimeStamp₁ is the minuend

TimeStamp₂ 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 the 48-bit universal LAN MAC addresses of IEEE 802, IEEE 802.1Q and to Table 15.

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 802.1AB (LLDP-PDU), IEEE 802.1AS (gPTP-PDU), PTCP-PDU	IEEE 802.1Q, chapter 6.1, 8.5, 8.13.2 and IEEE 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 802.1Q.

EXAMPLE 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]. The value of the field DestinationAddress shall be according to the 48-bit universal LAN MAC addresses of IEEE 802.

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

4.2.2.3.2 DCP-PDUs

For DCP-Multicast-PDUs, this field shall be coded according to Table 17, Table 18 and Table 19. 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

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-02 – 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 20, Table 21, and Table 22.

Table 20 – 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 21 – 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 22 – 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 23, Table 24, Table 25, Table 26, Table 27, and Table 28.

Table 23 – 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 24 – 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 25 – 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 26 – 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 27 – 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 28 – 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 29, Table 30, Table 31, and Table 32 show additional Multicast MAC address ranges according to this document.

Table 29 – 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 30 – 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 31 – MulticastMACAdd range 10

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

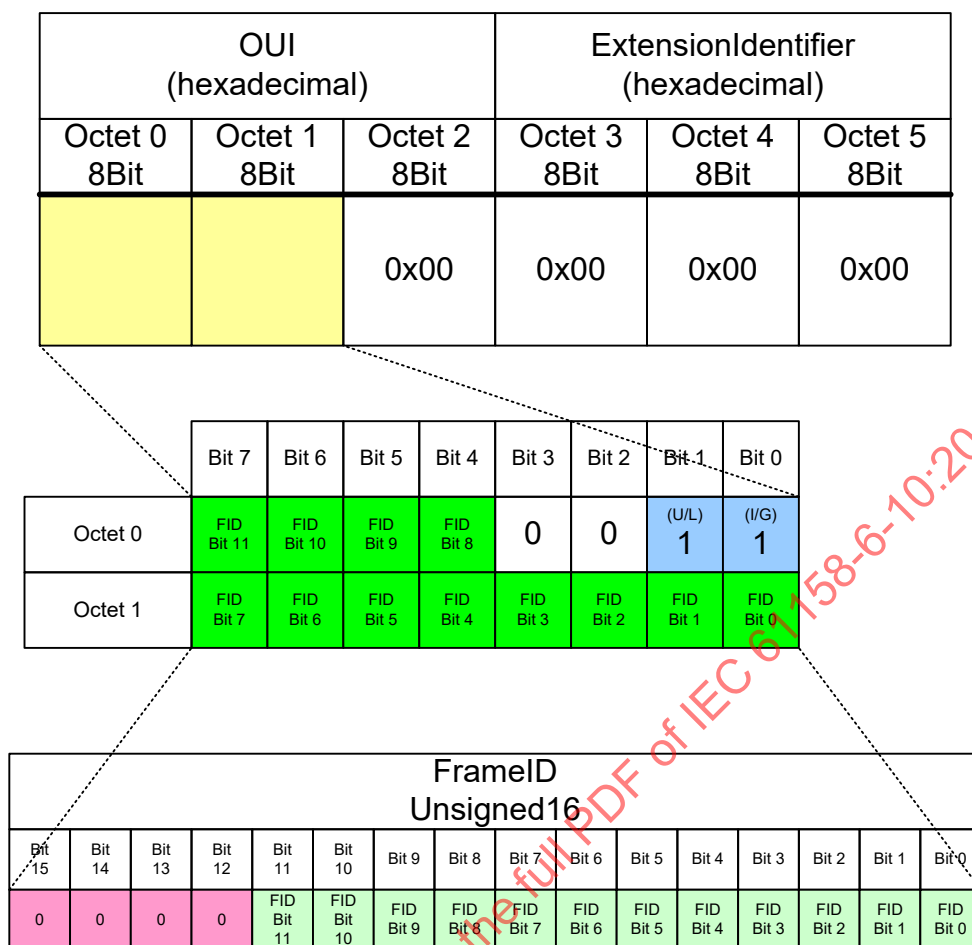
Table 32 – MulticastMACAdd range 11

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 802.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019



where

(U/L) means „Universally or Locally administered address“

(I/G) means „Individual/Group address“

FID means FrameID

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 33.

Table 33 – 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 34.

Table 34 – 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

4.2.2.4 Coding of the field LT

This field shall be coded as data type Unsigned16 with the values according to IEEE 802.3. This specification uses the values according to Table 35.

Table 35 – LT (Length/Type)

Value (hexadecimal)	Meaning
0x0800	IP (UDP, RPC, SNMP, ICMP)
0x0806	ARP
0x8100	Tag Control Information
0x8892	RTC, RTA, DCP, PTP, FRAG
0x88F7	IEEE 802.1AS
0x88CC	LLDP
0x88E3	MRP

4.2.2.5 Coding of the field TagControlInformation

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: TagControlInformation.VID

This field shall be coded according to IEEE 802.1Q with the values according to Table 36. This document does not define any VID.

Table 36 – TagControlInformation.VID

Value (hexadecimal)	Meaning
0x000	Default According to IEEE 802.1Q definition, No VLAN assigned
0x001 – 0xFFF	IEEE 802.1Q definition applies

Bit 12: TagControlInformation.DEI

This field shall be coded according to IEEE 802.1Q with the values according to Table 37.

Table 37 – TagControlInformation.DEI

Value (hexadecimal)	Meaning
0x00	Default Drop Eligible is false
0x01	Reserved Drop Eligible is true

Bit 13 – 15: TagControlInformation.PCP

This field shall be coded according to IEEE 802.1Q with the values according to Table 38.

Table 38 – TagControlInformation.PCP

Value (hexadecimal)	Meaning
0x00	IP, DCP
0x01	—
0x02	Recommended for IP PDUs containing PROFINETIOServiceReqPDU or PROFINETIOServiceResPDU
0x03	—
0x04	—
0x05	RTA_CLASS_UDP or RTA_CLASS_1 (High and Low prior)
0x06	RT_CLASS_UDP, RT_CLASS_1, RT_CLASS_2, RT_CLASS_3 ^a RTA_CLASS_UDP or RTA_CLASS_1 (High prior – optional)
0x07	PTCP-AnnouncePDU
^a RT_CLASS_3 frames uses TagControlInformation.PCP only outside of the RED period.	

NOTE If the used IP implementation does not provide Tag Control Information then the IP frames may be transmitted without Tag Control Information.

4.2.2.6 Coding of the field FrameID

This field shall be coded as data type Unsigned16 with the values according to Table 39, Table 40, Table 41, Table 42, Table 43, Table 44, Table 45, Table 46, Table 47, Table 48, Table 49, Table 50 and Table 51. This field identifies the structure and the type of the APDU.

Table 39 – 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 40 – 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 41 – FrameID range 3

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
0x1000 – 0x47FF	Reserved	—

^a A network monitor may sort “normal” and “DFP” frame by using the first SFCRC16 of the frame.

^b In this case two FrameIDs (for example 0xFFFF0 and 0xFFFF1) 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 42 – FrameID range 4

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

Table 43 – FrameID range 5

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

Table 44 – FrameID range 6

Value (hexadecimal)	Meaning	Use
0x8000 – 0xBBFF	Dedicated to communication class RT_CLASS_1 (GREEN) unicast	GREEN, non redundant, normal ^{a,b}
0xBC00 – 0xBFFF	Dedicated to RT_CLASS_1 (GREEN) multicast	GREEN, non redundant, normal ^{a,b}
^a Used as substitution for the former communication class RT_CLASS_1(legacy) shown in Table 45.		
^b May be used as RT_CLASS_2 (ORANGE) only in conjunction with ARProperties.StartupMode == Legacy		

Table 45 – FrameID range 7

Value (hexadecimal)	Meaning	Use
0xC000 – 0xF7FF	Dedicated to communication class RT_CLASS_UDP unicast	RT_CLASS_UDP (recommended) and RT_CLASS_1(legacy) ^a ; The concurrent use of a FrameID from this range for RT_CLASS_UDP and RT_CLASS_1 shall be rejected.
0xF800 – 0xFBFF	Dedicated to communication class RT_CLASS_UDP multicast	RT_CLASS_UDP (recommended) and RT_CLASS_1(legacy) ^a ; The concurrent use of a FrameID from this range for RT_CLASS_UDP and RT_CLASS_1 shall be rejected.
^a RT_CLASS_1(legacy) should be supported by IO controller and IO supervisor together with ARProperties.StartupMode:= Legacy. It should not be generated by an IO device.		

Table 46 – FrameID range 8

Value (hexadecimal)	Meaning	Use
0xFC00	Reserved	—
0xFC01	Alarm_High	RTA_CLASS_1 and RTA_CLASS_UDP; The concurrent use of this FrameID for RTA_CLASS_UDP and RTA_CLASS_1 shall be supported.
0xFC02 – 0xFDFF	Reserved	—
0xFE00	Reserved	—
0xFE01	Alarm_Low	RTA_CLASS_1 and RTA_CLASS_UDP; The concurrent use of this FrameID for RTA_CLASS_UDP and RTA_CLASS_1 shall be supported.
0xFE02 – 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 47 – FrameID range 9

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 48 – FrameID range 10

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

Table 49 – FrameID range 11

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

Table 50 – FrameID range 12

Value (hexadecimal)	Meaning	Use
0xFF80 – 0xFF8F	FragmentationFrameID See 4.2.2.7	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 51 – FrameID range 13

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

4.2.2.7 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 to identify the fragmented frame by the receiver and shall be coded according to Table 52.

Table 52 – 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 53.

Table 53 – FragmentationFrameID.Constant

Value (hexadecimal)	Meaning
0xFF8	Upper part of the FrameID according to Table 50

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 54 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 54 – DCP APDU syntax

APDU name	APDU structure
DCP-PDU	DCP-Multicast-PDU ^ DCP-Unicast-PDU
DCP-Multicast-PDU	DCP-Identify-ReqPDU ^g ^ 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 ^j , [NameOfStationBlock] ^ [AliasNameBlock], [IdentifyReqBlock] ^{* a}
DCP-IdentifyAll-ReqPDU	DCP-MC-Header, AllSelectorBlock
DCP-Hello-ReqPDU	DCP-UC-Header, NameOfStationBlockRes, IPPParameterBlockRes, DeviceIDBlockRes, [DeviceOptionsBlockRes ^ DeviceVendorBlockRes] ^e , DeviceRoleBlockRes, DeviceInitiativeBlockRes
DCP-Identify-ResPDU	DCP-UC-Header, { IdentifyResBlock ^{* b} , NameOfStationBlockRes ^c , IPPParameterBlockRes ^c , DeviceIDBlockRes ^c , DeviceVendorBlockRes ^c , DeviceOptionsBlockRes ^c , DeviceRoleBlockRes ^c , [DeviceInitiativeBlockRes] ^d , [DeviceInstanceBlockRes], [OEMDeviceIDBlockRes] } ^f
DCP-Get-ReqPDU	DCP-UC-Header, GetReqBlock [*]
DCP-Get-ResPDU	DCP-UC-Header, (GetResBlock ^ GetNegResBlock) ^{* i}
DCP-Set-ReqPDU	DCP-UC-Header, [StartTransactionBlock, BlockQualifier], SetResetReqBlock ^h ^ SetReqBlock [*] , [StopTransactionBlock, BlockQualifier]

APDU name	APDU structure
DCP-Set-ResPDU	DCP-UC-Header, (SetResBlock ^ SetNegResBlock)*
a	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 responded with a DCP-Identify-ResPDU if all filter criteria match.
b	The content of the field value of the IdentifyResBlock of the PDU shall contain the option and suboption requested in the IdentifyReqBlock. In case of AliasNameBlock only the one used in the DCP-IdentifyFilter-ReqPDU shall be responded.
c	This field shall only be present if it is not already part of the IdentifyResBlock and may be omitted if empty.
d	This field shall only be present if the usage of DCP-Hello-ReqPDU is activated.
e	This field should be omitted by the sender.
f	Additional blocks may be appended by the sender.
g	May be used in conjunction with interface MAC address.
h	If StartTransaction is used together with FactoryResetBlock or ResetToFactoryBlock, StopTransaction shall be part of the PDU. StopTransaction may be used without StartTransaction.
i	If Options/Suboptions requested by DCP-Get-ReqPDU don't fit into DCP-Get-ResPDU than, 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.
j	At least one of the optional blocks shall be used.

Table 55 defines structures for substitutions of elements of the APDU structures shown in Table 54.

Table 55 – 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
IdentifyResBlock	NameOfStationBlockRes ^ DeviceRoleBlockRes ^ DeviceVendorBlockRes ^ DeviceIDBlockRes ^ DeviceOptionsBlockRes ^ OEMDeviceIDBlockRes ^ MACAddressBlockRes ^ IPParameterBlockRes ^ DHCPParameterBlockRes ^ ManufacturerSpecificParameterBlockRes ^ AliasNameBlockRes
GetReqBlock	MACAddressType ^ IPParameterType ^ FullIPSuiteType ^ DeviceVendorType ^ NameOfStationType ^ DeviceIDType ^ DeviceRoleType ^ DeviceOptionsType ^ DeviceInstanceType ^ OEMDeviceIDType ^ StandardGatewayType ^ DHCPParameterType ^ DeviceInitiativeType ^ ManufacturerSpecificParameterType
GetResBlock	MACAddressBlockRes ^ IPParameterBlockRes ^ FullIPSuiteBlockRes ^ DeviceVendorBlockRes ^ NameOfStationBlockRes ^ DeviceIDBlockRes ^ DeviceRoleBlockRes ^ DeviceOptionsBlockRes ^ DeviceInstanceBlockRes ^ OEMDeviceIDBlockRes ^ StandardGatewayBlockRes ^ DHCPParameterBlockRes ^ DeviceInitiativeBlockRes ^ ManufacturerSpecificParameterBlockRes
GetNegResBlock	ControlOption, SuboptionResponse, DCPBlockLength, MACAddressType ^ IPParameterType ^ FullIPSuiteType ^ DeviceVendorType ^ NameOfStationType ^ DeviceIDType ^ DeviceRoleType ^ DeviceOptionsType ^ DeviceInstanceType ^ OEMDeviceIDType ^ StandardGatewayType ^ DHCPParameterType ^ DeviceInitiativeType ^ ManufacturerSpecificParameterType, BlockError, [AddDataValue]

Substitution name	Structure
SetReqBlock	IPParameterType ^ FullIPSuiteType ^ NameOfStationType ^ DHCPParameterType ^ SignalType ^ ManufacturerSpecificParameterType, DCPBlockLength, BlockQualifier, IPParameterValue ^ FullIPSuiteValue ^ NameOfStationValue ^ DHCPParameterValue ^ SignalValue ^ ManufacturerSpecificParameterValue 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, BlockError(=0), [AddDataValue]
SetNegResBlock	ControlOption, SuboptionResponse, DCPBlockLength, IPParameterType ^ FullIPSuiteType ^ NameOfStationType ^ DHCPParameterType ^ StartTransactionType ^ StopTransactionType ^ SignalType ^ FactoryResetType ^ ResetToFactoryType ^ ManufacturerSpecificParameterType, 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
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

Substitution name	Structure
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 64 for the values ^b See Table 65, Table 66, Table 67, Table 68, Table 69, Table 70, Table 71, and Table 72 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
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".
DHCPPParameterType	DHCPOption, SuboptionDHCP
DHCPPParameterBlock	DHCPPParameterType, DCPBlockLength, DHCPPParameterValue
DHCPPParameterBlockRes	DHCPPParameterType, DCPBlockLength, BlockInfo, DHCPPParameterValue
DHCPPParameterValue	SuboptionDHCP, DHCPPParameterLength, DHCPPParameterData
ManufacturerSpecificParameterType	ManufacturerSpecificOption, SuboptionManufacturerSpecific
ManufacturerSpecificParameterBlock	ManufacturerSpecificParameterType, DCPBlockLength, ManufacturerSpecificParameterValue
ManufacturerSpecificParameterBlockRes	ManufacturerSpecificParameterType, DCPBlockLength, BlockInfo, ManufacturerSpecificParameterValue
ManufacturerSpecificParameterValue	ManufacturerOUI, ManufacturerSpecificString

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 56. They shall be set in the appropriate PDU.

Table 56 – ServiceID

Value (hexadecimal)	Meaning
0x00 – 0x02	Reserved
0x03	Get
0x04	Set
0x05	Identify
0x06	Hello
0x07 – 0xFF	Reserved

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 57.

Table 57 – 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 58.

Table 58 – 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 59.

Table 59 – 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 60.

Table 60 – 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 61.

Table 61 – 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 62.

Table 62 – 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 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

This field shall be coded as data type Unsigned16 according to Table 63. It shall contain a delay factor that shall be used by the server to calculate its individual response delay time.

Table 63 – ResponseDelayFactor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Allowed value without spread
0x0002 – 0x1900	Allowed value with spread
0x1901 – 0xFFFF	Reserved

The server shall calculate the response delay time (allowed values between 0 ms and 64 s) according to the following formulae.

The last two octets of the responder MAC address (see the 48-bit universal LAN MAC addresses of IEEE 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.

$$\text{Spread} = K \text{ MOD } \text{ResponseDelayFactor} \quad (4)$$

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:

$$\text{Minimum Response Delay} = 10 \text{ ms} \times \text{Spread} \quad (5)$$

where

Minimum Response Delay	is the minimal response delay
Spread	is the specific spreading factor

Spread equals zero:

$$\text{Minimal Response Delay} = 0 \text{ ms} \quad (6)$$

where

Minimal Response Delay	is the minimal response delay

The client shall calculate the response delay timeout according to the following formulae.

ResponseDelayFactor equals one:

$$\text{Response delay timeout} = 400 \text{ ms} \quad (7)$$

where

Response delay timeout is the response delay timeout

ResponseDelayFactor is greater than one:

$$\text{Intermediate value} = 1 \text{ s} + \text{ResponseDelayFactor} \times 10 \text{ ms} \quad (8)$$

where

Intermediate value is the intermediate value of the response delay

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

$$\text{Response delay timeout} = \text{Round up to next second (intermediate value)} \quad (9)$$

where

Response delay timeout is the response delay timeout

Intermediate value is the intermediate value of the response delay

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 64 shows the list of available options and Table 65, Table 66, Table 67, Table 68, Table 69, Table 70 and Table 71 show the list of available suboptions.

Table 64 – List of options

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

Table 65 – List of suboptions for option IPOption

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

Table 66 – List of suboptions for option DevicePropertiesOption

Value (hexadecimal)	Meaning	Accessible
0x01	SuboptionDeviceVendor	Read
0x02	SuboptionNameOfStation	Read / Write
0x03	SuboptionDeviceID	Read
0x04	SuboptionDeviceRole	Read
0x05	SuboptionDeviceOptions	Read
0x06	SuboptionAliasName	Used as filter only
0x07	SuboptionDeviceInstance	Read Optional
0x08	SuboptionOEMDeviceID	Read Optional
0x09	SuboptionStandardGateway	Read Optional ^a
Other	Reserved	—

^a Shall be supported if StandardGatewayValue.StandardGateway == 0x01

Table 67 – List of suboptions for option DHCPOption

Value (hexadecimal)	Meaning	Accessible
See Table 72	SuboptionDHCP	Read / Write Optional

Table 68 – List of suboptions for option ControlOption

Value (hexadecimal)	Meaning	Accessible
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 69 – List of suboptions for option DeviceInitiativeOption

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

Table 70 – List of suboptions for option AllSelectorOption

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

Table 71 – List of suboptions for option ManufacturerSpecificOption

Value (hexadecimal)	Meaning	Accessible
0x00 – 0xFF	SuboptionManufacturerSpecific	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 64.

4.3.1.4.2.2 Coding of the field SuboptionMACAddress

This field shall be coded as data type Unsigned8 according to Table 64.

4.3.1.4.2.3 Coding of the field SuboptionIPParameter

This field shall be coded as data type Unsigned8 according to Table 64.

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 64.

4.3.1.4.3.2 Coding of the field SuboptionDeviceVendor

This field shall be coded as data type Unsigned8 according to Table 66.

4.3.1.4.3.3 Coding of the field SuboptionNameOfStation

This field shall be coded as data type Unsigned8 according to Table 66.

4.3.1.4.3.4 Coding of the field SuboptionDeviceID

This field shall be coded as data type Unsigned8 according to Table 66.

4.3.1.4.3.5 Coding of the field SuboptionDeviceRole

This field shall be coded as data type Unsigned8 according to Table 66.

4.3.1.4.3.6 Coding of the field SuboptionDeviceOptions

This field shall be coded as data type Unsigned8 according to Table 66.

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 66.

4.3.1.4.3.8 Coding of the field SuboptionDeviceInstance

This field shall be coded as data type Unsigned8 according to Table 66.

4.3.1.4.3.9 Coding of the field SuboptionOEMDeviceID

This field shall be coded as data type Unsigned8 according to Table 66.

4.3.1.4.3.10 Coding of the field SuboptionStandardGateway

This field shall be coded as data type Unsigned8 according to Table 66.

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 64.

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 72.

Table 72 – SuboptionDHCP

Value (decimal)	Description	References	Accessible
0 – 11	Reserved ^a	—	—
12	Host Name ^d	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 ^d	—	Read / Write Optional
82 – 96	Reserved ^a	—	—
97	UUID-based Client (=addressed Station) Identifier	—	Read / Write Optional
98 – 254	Reserved ^a	—	—
255 ^b	See 4.3.1.4.22	—	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 By IETF RFC 2132 defined as "END" but handled internally. Thus this key is overloaded as DHCP control in this document.

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

^d The use of these suboptions is still not fixed and subject to change in later revisions.

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 64.

4.3.1.4.5.2 Coding of the field SuboptionStart

This field shall be coded as data type Unsigned8 according to Table 68 and with a DCPBlockLength according to Table 73.

Table 73 – Coding of DCPBlockLength in conjunction with SuboptionStart

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

There are 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 68 and with a DCPBlockLength according to Table 74.

Table 74 – Coding of DCPBlockLength in conjunction with SuboptionStop

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

There are 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 68 and with a DCPBlockLength according to Table 75.

Table 75 – 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 68.

4.3.1.4.5.6 Coding of the field SuboptionFactoryReset

This field shall be coded as data type Unsigned8 according to Table 68 and with a DCPBlockLength according to Table 76.

Table 76 – Coding of DCPBlockLength in conjunction with SuboptionFactoryReset

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

There are no additional data for this suboption.

Table 77 shows the alignment between FactoryReset and ResetToFactory.

Table 77 – 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 with 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 68 and with a DCPBlockLength according to Table 78.

This Suboption offers different modes coded by the BlockQualifier with the meaning shown in Table 79.

Table 78 – Coding of DCPBlockLength in conjunction with SuboptionResetToFactory

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

There are no additional data for this suboption.

Table 79 shows the behavior of the reset command for the different modes.

Table 79 – Meaning of the different ResetToFactory modes

Affected object	Interface				Device	
	Mode 1	Mode 2	Mode 3	Mode 4	Mode 8	Mode 9
NameOfStation / NameOfInterface	—	NULL	NULL	NULL	NULL	NULL
IP-suite	—	No IP-suite	No IP-suite	No IP-suite	No IP-suite	No IP-suite
Persistent PDev records	—	NULL	NULL	NULL	NULL	NULL
I&M data	NULL	—	NULL	NULL	NULL	NULL
SNMP writeable OIDs	NULL	— ^a	NULL	NULL	NULL	NULL
Persistent records (not PDev)	NULL	—	NULL	NULL	NULL	NULL
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
^a The expected behavior differs between single and multiple interface devices. Single interface: Should be deleted (NULL) Multiple interface: May be deleted (NULL)						

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 64.

4.3.1.4.6.2 Coding of the field SuboptionDeviceInitiative

This field shall be coded as data type Unsigned8 according to Table 69 and with a DCPBlockLength according to Table 80.

Table 80 – 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 AllSelectorOption

4.3.1.4.7.1 Coding of the field AllSelectorOption

This field shall be coded as data type Unsigned8 according to Table 64.

4.3.1.4.7.2 Coding of the field SuboptionAllSelector

This field shall be coded as data type Unsigned8 according to Table 70.

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 64.

4.3.1.4.8.2 Coding of the field SuboptionManufacturerSpecific

This field shall be coded as data type Unsigned8 according to Table 71.

4.3.1.4.9 Coding of the field DCPBlockLength

This field shall be coded as data type Unsigned16 according to Table 81.

Table 81 – 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

4.3.1.4.10 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 82.

Table 82 – 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.

Allowed values with the option ControlOption and suboption SuboptionResetToFactory shall be according to Table 83.

Table 83 – 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 have been stored permanent 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 if permanently stored reset. This mode is intended to set the addressed communication interface of a device into a state which almost similar to the “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 its factory values.	
	4	Recommended Reset all stored data Reset all stored data to its factory default values.	
	5 – 7	Reserved	
	8	Recommended Reset device Reset all stored data to its factory values. This service shall reset the communication parameters of all interfaces of the device and should reset all parameters of the device. This mode is intended to set the device into a state which is similar to the “out of the box” state.	Device
	9	Optional Reset and restore data Reset installed software revisions to factory images.	
	10 – 15	Reserved	
	other	Reserved	—

Allowed values with the option ControlOption and NOT suboption SuboptionResetToFactory shall be according to Table 84.

Table 84 – BlockQualifier with option ControlOption and NOT suboption SuboptionResetToFactory

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

4.3.1.4.11 Coding of the field BlockError

This field shall be coded as data type Unsigned8 and shall be set according to Table 85.

Table 85 – 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	SET not possible by local reasons	If Set service is not possible for local reasons
0x06	In operation, SET 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.

4.3.1.4.12 Coding of the field BlockInfo

This field shall be coded as data type Unsigned16. The allowed values shall be set according to Table 86, Table 87, Table 88 and Table 89.

Table 86 – BlockInfo for SuboptionIPParameter

Bit	Meaning
Bit 0 – 1	See Table 87
Bit 2 – 6	Reserved
Bit 7	See Table 88
Bit 8 – 15	Reserved

Table 87 – 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 88 – 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 local 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 89 – BlockInfo for all other suboptions

Bit	Meaning
Bit 0 – 15	Reserved

4.3.1.4.13 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 90.

Table 90 – 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.

4.3.1.4.14 Coding of the field SignalValue

This field shall be coded as data type Unsigned16 according to Table 91.

Table 91 – SignalValue

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

4.3.1.4.15 Coding of the field NameOfStationValue

4.3.1.4.15.1 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.

4.3.1.4.15.2 Encoding

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

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

- 1 or more labels, separated by [.]
- Total length is 1 to 240
- Label length is 1 to 63
- Labels consist of [a-z0-9-]
- Labels do not start with [-]
- Labels do not end with [-]
- Labels do not use multiple concatenated [-] except for IETF RFC 5890
- 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 The field NameOfStationValue is not terminated by zero.

4.3.1.4.15.3 Semantics with Identify service

The following rules shall be applied:

- Name of station with length 1 to 240 octets:
Only the device shall answer which has exactly got this name. In the comparison of the two names on 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 4.3.1.4.15.2.

4.3.1.4.15.4 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 4.3.1.4.15.2.

4.3.1.4.16 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 65535. 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 values “port-001-00000” shall be used for the first port submodule for an interface within slot 0 if any other slot may contain another port submodule.

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

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

4.3.1.4.17 Coding of the field AliasNameValue

4.3.1.4.17.1 Encoding

This field shall be coded as OctetString. The content shall be the concatenation of the content 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

4.3.1.4.17.2 Semantics with Identify service

The following rules shall be applied:

- Alias name with length != 0:
Only the device shall answer which has exactly got this additional alias name. In the comparison of the two names on 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 get through means of DCP.

4.3.1.4.18 Coding of the field DeviceRoleDetails

This field shall be coded as data type Unsigned8 with the values according to Table 92.

Table 92 – DeviceRoleDetails

Bit	Meaning	Usage
Bit 0	PNIO Device	The device contains an IO device interface.
Bit 1	PNIO Controller	The device contains an IO controller interface.
Bit 2	PNIO Multidevice	The device contains multiple IO device interfaces.
Bit 3	PNIO Supervisor	The device contains an IO supervisor interface.
Bit 4 – 7	Reserved	—

4.3.1.4.19 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 802.

4.3.1.4.20 Coding section related to IPSuite

4.3.1.4.20.1 General

The rules for the fields in 4.3.1.4.20 differ between one interface and multiple (more than one) interface devices.

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

4.3.1.4.20.2 Coding of the field IPAddress

4.3.1.4.20.2.1 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 to Table 93.

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

Table 93 – 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"

Value (CIDR ^a)	Comment	Range	Meaning
Reserved	Invalid	—	Host part of the IPAddress is a series of consecutive “1” (subnet broadcast address) IPAddress may be accepted but should be invalid.
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.

4.3.1.4.20.2.2 Single interface device

Check defined in Table 93 applies.

4.3.1.4.20.2.3 Multiple interface device

Check defined in Table 93 applies per interface.

However the IP stack within the device has to have an unambiguous routing decision for outgoing packets.

If the device supports IETF RFC 950, the routing table will sort routing entries in reverse order according to the length of their subnet mask (i.e. 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 may overlap in this case.

Otherwise, the subnets of the interfaces shall not overlap, i.e. for two interfaces x and y, the equation “(IFx_IPAddress & IFx_SubnetMask) != ((IFy_IPAddress & IFy_SubnetMask) & IFx_SubnetMask)” applies in each case.

In case the IPAddress range is not ambiguous, 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.

4.3.1.4.20.3 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 94.

Table 94 – 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

4.3.1.4.20.4 Coding of the field StandardGateway**4.3.1.4.20.4.1 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 93 and Table 95.

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

Table 95 – 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 ^d	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 ^b	Standard gateway assigned Should be checked against the field subnetmask
Reserved ^c	No standard gateway assigned
^a Notation according to classless inter domain routing defined in IETF RFC 4632 ^b Valid according to Table 93 and Table 94 ^c Invalid according to Table 93 and Table 94 ^d Should only be used for nodes with routing support. For nodes without routing support only 0.0.0.0 should be used.	

4.3.1.4.20.4.2 Single interface device

Check defined in Table 95 applies.

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

4.3.1.4.20.4.3 Multiple interface device

Check defined in Table 95 applies per interface.

An assigned StandardGateway has to 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 (including also those interfaces which are not configured via this specification, i.e. IP only interfaces).

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

- If no gateway is assigned, there is no effective gateway. Any existing diagnosis information related to the gateway must be cleared.

No means that no interface has an assigned StandardGateway.

- If exactly one StandardGateway is assigned, then this is the effective gateway and its value is applied to the IP stack. In case a “StandardGateway mismatch” diagnosis was indicated before, it must be cleared.

Exactly one means that at least one interface must 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 over ExtChannelAddValue the currently effective StandardGateway. Whether an effective StandardGateway is elected in this case and which was chosen is 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 over ExtChannelAddValue the currently inactive StandardGateway.

4.3.1.4.20.5 Correlations between the subfields of IPSuite

Table 96 shows the correlation between Table 93, Table 94, and Table 95.

Table 96 – Correlation between the subfields of IPSuite

Table 93 value	Table 94 value	Table 95 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			

4.3.1.4.21 Coding of the fields DHCPParameterValue using DHCP Option 61

4.3.1.4.21.1 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:

4.3.1.4.21.2 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.

4.3.1.4.21.3 Use of MACAddress as client identifier

If the device shall obtain its IP parameters using the DHCP protocol and use the MAC address as a client identifier, then the option 61 shall have the values according Table 97.

Table 97 – MACAddress as client identifier

Parameter name	Parameter value (decimal)
SuboptionDHCP	61
DHCPPParameterLength	1
DHCPPParameterData	1

4.3.1.4.21.4 Use of NameOfStation as client identifier

If the device shall obtain its IP parameters using the DHCP protocol and use the NameOfStation as client identifier the option 61 shall have the values according to Table 98 (in addition to IETF RFC 2132).

Table 98 – NameOfStation as client identifier

Parameter name	Parameter value (decimal)
SuboptionDHCP	61
DHCPPParameterLength	1
DHCPPParameterData	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.

4.3.1.4.21.5 Use of arbitrary client identifier

If the device shall obtain its IP parameters using the DHCP protocol and use an arbitrary string as client identifier the option 61 shall have the values according Table 99.

Table 99 – Arbitrary client identifier

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

4.3.1.4.22 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 overloaded by DCP and shall be used according to Table 100.

Table 100 – DHCPParameterValue using DHCP Option 255

Parameter name	Parameter value (decimal)
SuboptionDHCP	255
DHCPParameterLength	1
DHCPParameterData	0: Don't use DHCP (Default) 1: Don't use DHCP but set all DHCPOptions to their "Reset to factory" value 2: 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.

4.3.1.4.23 Coding of the field ManufacturerOUI

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

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

4.3.1.4.24 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.

4.3.1.4.25 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.

4.3.1.4.26 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.

4.3.1.4.27 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.

4.3.1.4.28 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 101.

Table 101 – 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.

4.3.2 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 102.

Table 102 – 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	—

Primitive	Source	Destination	Associated parameters	Functions
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 primitives including their associated parameters issued or received by DCPUCS are described in the service definition and shown in Table 103.

Table 103 – 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 28.

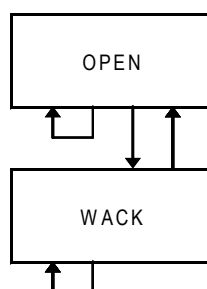


Figure 28 – State transition diagram of DCPUCS

States of the DCPUCS

- 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 doesn't respond, retry the request and after the retry limit is 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. Within the OPEN state, DCP Get and Set service requests issued by the DCP user are transformed to Data Link Layer services. After issuing the transmission of data, the WACK state is used to wait for the Response-PDU causing a confirmation primitive to the DCP user.

4.3.2.2.1.4 DCPUCS state table

Table 104 contains the complete description of the DCPUCS state table.

Table 104 – 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
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

#	Current State	Event /Condition =>Action	Next State
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
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

#	Current State	Event /Condition =>Action	Next State
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 105 contains the functions, macros, timers and variables used by the DCPUCS and their arguments and their descriptions.

Table 105 – 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 is expired.
UC Client Timer	Timer	This Timer, restarted on the requestor when sending a 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 service primitives including their associated parameters issued or received by DCPUCR are described in the service definition and shown in Table 106.

Table 106 – Remote primitives issued or received by DCPUCR

Primitive	Source	Destination	Associated parameters	Functions
DCP_Set.rsp	CMINA	DCPUCR	CREP, ListOfResponse	—
LMPM_A_Data.cnf	LMPM	DCPUCR	CREP, LMPM_status	—
LMPM_A_Data.ind	LMPM	DCPUCR	CREP, DA, SA, Prio, A_SDU	—
DCP_Set.ind	DCPUCR	CMINA	CREP, ListOfData, ListOfControlCommands	—
LMPM_A_Data.req	DCPUCR	LMPM	CREP, DA, SA, Prio, A_SDU	—

4.3.2.2.1.2 Primitives exchanged between local machines

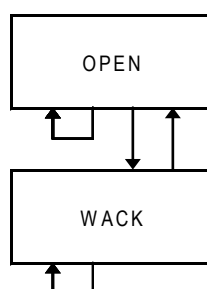
The local primitives including their associated parameters issued or received by DCPUCR are described in the service definition and shown in Table 107.

Table 107 – Local primitives issued or received by DCPUCR

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.3.2.2.2 State transition diagram

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

**Figure 29 – State transition diagram of DCPUCR**

States of the DCPUCR are:

- OPEN** Wait for a DCP Get or Set service and prepare the response for GET
- WACK** Wait for the application response for SET to create the DCP set response

4.3.2.2.3 State machine description

Within the OPEN state, the state machine is waiting for the first LMPM_A_Data service indication. After passing the indication to the user the WACK state is entered. A response service will activate an LMPM_A_Data request and bring the machine back to the OPEN state.

4.3.2.2.4 DCPUCR state table

Table 108 contains the complete description of the DCPUCR state table. A LMPM_A_Data.ind primitive will be accepted if type is DCP-Get-ResPDU or DCP-Set-ResPDU.

Table 108 – DCPUCR state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	DCP_Set.rsp () => ignore	OPEN
2	OPEN	LMPM_A_Data.ind () /DCP-Get-ReqPDU && (SAM == NIL SAM == SA) && (DA != Multicast) && ValidSelectors => SAM:= SA SXID:= XID SDA:= SA DA:= SDA XID:= SXID A_SDU:= CreateDCP-Get-ResPDU(ListOfData) StartTimer (Client Hold Time) LMPM_A_Data.req ()	OPEN
3	OPEN	LMPM_A_Data.ind () /DCP-Get-ReqPDU && (SAM == NIL SAM == SA) && (DA != Multicast) && !ValidSelectors => SAM:= SA SXID:= XID SDA:= SA DA:= SDA XID:= SXID ReturnCodeFromValidSelectors A_SDU:= Create DCP-Get-ResPDU with error information StartTimer (Client Hold Time) LMPM_A_Data.req ()	OPEN
4	OPEN	LMPM_A_Data.ind () /DCP-Set-ReqPDU && (SAM == NIL SAM == SA) && (DA != Multicast) && ValidListOfData => SAM:= SA SXID:= XID SDA:= SA DCP_Set.ind (CREP, ListOfData)	WACK

#	Current State	Event /Condition =>Action	Next State
5	OPEN	LMPM_A_Data.ind () /DCP-Set-ReqPDU && (SAM == NIL SAM == SA) && (DA != Multicast) && !ValidListOfData => ignore	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.cnf () => ignore	OPEN
9	OPEN	TimerExpired (Client Hold Time) => SAM:= NIL	OPEN
10	WACK	DCP_Set.rsp () => DA:= SDA XID:= SXID A_SDU:= DCP-Set-ResPDU containing the response from the application StartTimer (Client Hold Time) LMPM_A_Data.req ()	OPEN
11	WACK	LMPM_A_Data.ind () => ignore	WACK
12	WACK	LMPM_A_Data.cnf () => ignore	WACK
13	WACK	TimerExpired (Client Hold Time) => ignore	WACK

4.3.2.2.2.5 Functions, Macros, Timers and Variables

Table 109 contains the functions, macros, timers and variables used by the DCPUCR and their arguments and their descriptions.

Table 109 – Functions, Macros, Timers and Variables used by the DCPUCR

Name	Type	Function/Meaning
CreateDCP-Get-ResPDU	Function	Create DCP-Get-ResPDU with ListOfData taken from the DCP ASE according to ValidSelectors from the DCP-Get-ReqPDU.
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 is issued if a timer expires
ListOfData	Macro	If the requested ValidSelectors doesn't fit into a DCP-Get-ResPDU, than fill into ListOfData as long as the requested Options/Suboptions fits in completely. Any leftover from ValidSelectors shall be skipped.
ReturnCodeFromValidSelectors	Macro	See Table 85
ValidListOfData	Macro	The PDU shall only pass this check if the response fits into the associated DCP-Set-ResPDU.
ValidSelectors	Macro	Check the PDU against the DCP definition and the DCP ASE
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 value of this time is by default 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.

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 service primitives including their associated parameters issued or received by DCPMCS are described in the service definition and shown in Table 110.

Table 110 – 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	—

Primitive	Source	Destination	Associated parameters	Functions
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 111.

Table 111 – 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 30.

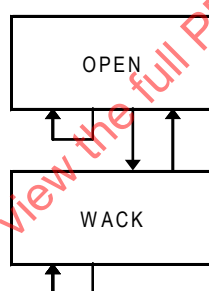


Figure 30 – State transition diagram of DCPMCS

States of the DCPMCS are:

- OPEN** Wait for an Identify service request.
- WACK** Wait for all responses until a certain 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 the data the WACK state is used to wait for all responses until a certain time. The timeout causes to issue the Identify confirmation service primitive with all received responses to the DCP user and sets the state machine to the OPEN state.

4.3.2.2.3.4 DCPMCS state table

Table 112 contains the complete description of the DCPMCS state machine. A LMPM_A_Data.ind primitive shall be accepted if type is DCP-IdentifyResPDU.

Table 112 – 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
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

#	Current State	Event /Condition =>Action	Next State
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 109 contains the functions, macros, timers and variables used by the DCPMCS and their arguments and their descriptions.

Table 113 – 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 the speed up of the connection establishing.
Response delay timer	Timer	This timer is used to build a 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 Primitive definitions

4.3.2.2.4.1.1 Primitives exchanged between remote machines

The service primitives including their associated parameters issued or received by DCPMCR are described in the service definition and shown in Table 114.

Table 114 – 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	—

4.3.2.2.4.1.2 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 115.

Table 115 – Local primitives issued or received by DCPMCR

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.3.2.2.4.2 State transition diagram

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

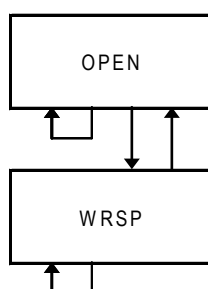


Figure 31 – State transition diagram of DCPMCR

States of the DCPMCR are:

OPEN	Waiting 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	Waits the response delay to send the response in case the DCP user has recognized a match with local data.

4.3.2.2.4.3 State machine description

The state machine is waiting 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 a certain time calculated by means of the clients Response Delay Factor to send the

response in case the DCP user has recognized a match with local data. Otherwise, nothing shall be sent.

4.3.2.2.4.4 DCPMCR state table

Table 116 contains the complete description of the DCPMCR state machine. A LMPM_A_Data.ind primitive shall be accepted if Type is DCP-IdentifyReqPDU.

Table 116 – 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) 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 A_SDU:= Create DCP-Identify-ResPDU with ListOfData taken from the DCP ASE according the ListOfFilters of the DCP-Identify-ReqPDU LMPM_A_Data.req ()	OPEN
6	WRSP	LMPM_A_Data.ind () => ignore	WRSP
7	WRSP	LMPM_A_Data.cnf () => ignore	WRSP

4.3.2.2.4.5 Functions, Macros, Timers and Variables

Table 117 contains the functions or macros used by the DCPMCR and their arguments and their descriptions.

Table 117 – 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
CalculateResponseDelay	Macro	This local macro calculates the ResponseDelay according 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 time-based 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 service primitives including their associated parameters issued or received by DCPHMCS are described in the service definition and shown in Table 118.

Table 118 – 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

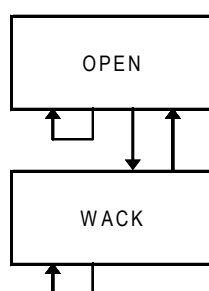
The local primitives including their associated parameters issued or received by DCPHMCS are described in the service definition and shown in Table 119.

Table 119 – 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 32.

**Figure 32 – 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 the data the WACK state is used to wait for the transmit confirmation.

4.3.2.2.5.4 DCPHMCS state table

Table 120 contains the complete description of the DCPHMCS state machine.

Table 120 – 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

#	Current State	Event /Condition =>Action	Next State
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 121 contains the functions, macros, timers and variables used by the DCPHMCS and their arguments and their descriptions.

Table 121 – 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 service primitives including their associated parameters issued or received by DCPHMCR are described in the service definition and shown in Table 122.

Table 122 – Remote primitives issued or received by DCPHMCR

Primitive	Source	Destination	Associated parameters	Functions
LMPM_A_Data.cnf	LMPM	DCPHMCR	CREP, LMPM_status	—
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 123.

Table 123 – 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 33.

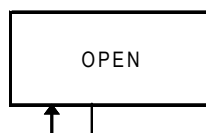


Figure 33 – State transition diagram of DCPHMCR

State of the DCPHMCR is:

OPEN

waiting for DCP-Hello-ReqPDUs

4.3.2.2.6.3 State machine description

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

4.3.2.2.6.4 DCPHMCR state table

Table 124 contains the complete description of the DCPHMCR state machine. A LMPM_A_Data.ind primitive will be accepted if Type is DCP-Hello-ReqPDU.

Table 124 – 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
3	OPEN	LMPM_A_Data.cnf () => ignore	OPEN

4.3.2.2.6.5 Functions, Macros, Timers and Variables

Table 125 contains the functions, macros, timers and variables used by the DCPHMCR and their arguments and their descriptions.

Table 125 – 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 working time control

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 126 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 126 – 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 127 defines structures for substitutions of elements of the APDU structures.

Table 127 – 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

Substitution name	Structure
PTCPortTime	PTCP_TLVHeader, [Padding*] ^a , PTCP_T2TimeStamp
PTCOption	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 according block.

Bit 9 – 15: PTCP_TLVHeader.Type

This field shall be coded with the values according to Table 128.

Table 128 – PTCP_TLVHeader.Type

Value (hexadecimal)	Meaning	Use
0x00	PTCPEnd	Mandatory
0x01	PTCPSubdomain	Mandatory
0x02	PTCPTime	Mandatory
0x03	PTCPTimeExtension	Mandatory
0x04	PTCPMaster	Mandatory
0x05	PTCPortParameter	Mandatory
0x06	PTCPDelayParameter	Mandatory
0x07	PTCPortTime	Mandatory
0x08 – 0x7E	Reserved	—
0x7F	PTCOption Organizationally specific	Optional

4.4.1.3.5 Coding section related to PTCP-Header

4.4.1.3.5.1 General

These fields contains the propagation delay time in nanoseconds. It is parted into a field for the 10 ns part and a field for the 1 ns part. All delay fields are only relevant for Sync-, FollowUp-, DelayRes- without DelayFuRes- 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 contains the 10 ns part of the propagation delay time. This field shall be coded with the values according to Table 129.

Table 129 – PTCP_Delay10ns

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	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 130.

Table 130 – 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 contains the 1 ns part of the propagation delay time. This field shall be coded with the values according to Table 131.

Table 131 – PTCP_Delay1ns

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	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 contains the correction of the propagation delay time in 1 ns. This field shall be coded with the values according to Table 132.

Table 132 – 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 133.

Table 133 – PTCP_SequenceID

Meaning	Usage
Sync frame	The PTCP master increments this field for every sync frame.
Follow up frame	The PTCP master or a follow up injecting 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 34.

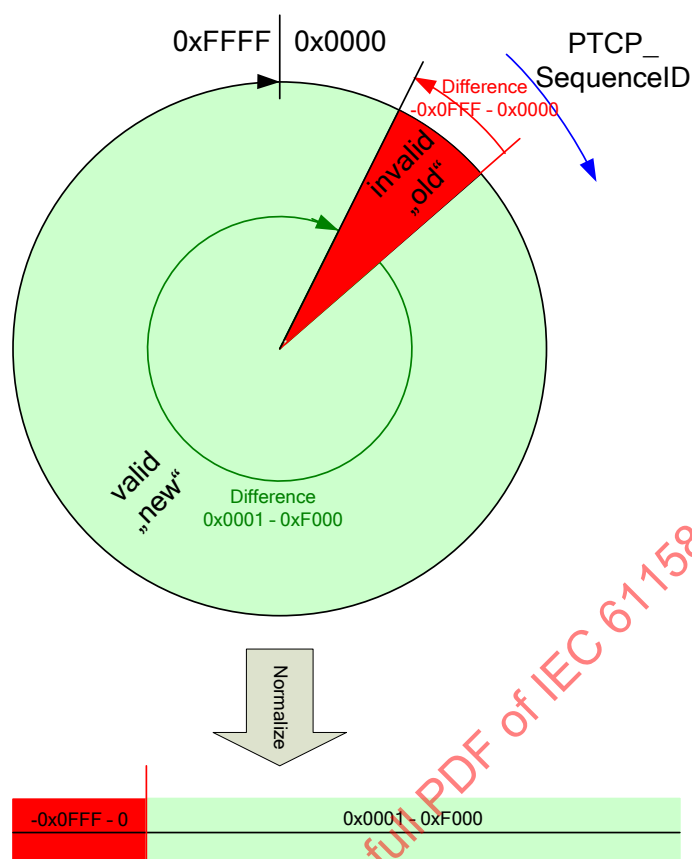


Figure 34 – PTCP_SequenceID value range

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 802.

NOTE The value of this central administrative number is given by the IEEE Registration Authority Committee. Available at <<http://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 organizationally specific. For the OUI the parameter PTCP_SubType shall be coded as in Table 134.

Table 134 – 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 135. The resolution of this field is 1 s.

Table 135 – 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 136. The resolution of this field is 1 ns.

Table 136 – 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 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 137.

Table 137 – 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 Table 138.

When used as WorkingClock, the timescale is arbitrary.

Table 138 – 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
0x00000000000010000	0	65	536 000 000	32
0x00000000100000000	0	4 294 967	296 000 000	32
0x00010000000000000	65	2 302 102 470	656 000 000	32

$$PTCP\_Epoch = PTCP\_EpochNumber \times 0x100000000 \quad (11)$$

where

PTCP_Epoch is the helper 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 helper variable for the calculation of seconds

PTCP_Second is the PTCP second value

PTCP_Nanosecond is the PTCP nanosecond value

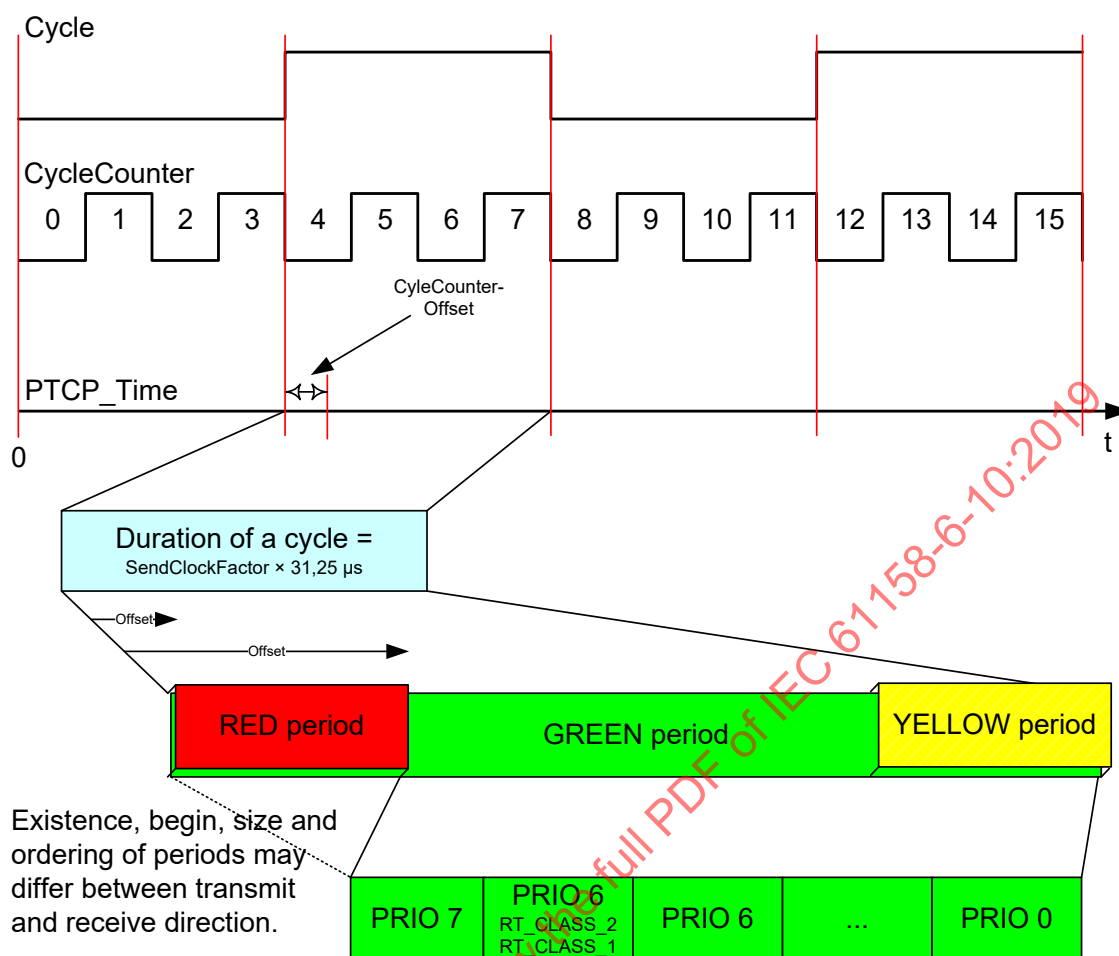


Figure 35 – Timescale correspondence between PTCP_Time and CycleCounter

For the synchronization of the CycleCounter as shown in Figure 35, the calculation shall accept PTCP_Time values with an offset to the cycle begin.

$$\text{Cycle} = \text{PTCP\_Time} \text{ DIV } (\text{SendClockFactor} \times 31,25 \mu\text{s}) \quad (13)$$

where

Cycle is the cycle
PTCP_Time is the PTCP time
SendClockFactor is the factor of the send clock, the length of one phase

$$\text{CycleCounter} = \text{Cycle} \times \text{SendClockFactor} \quad (14)$$

where

CycleCounter is the counter of cycles
Cycle is the cycle
SendClockFactor is the factor of the send clock, the length of one phase

$$\text{CycleCounterOffset} = \text{PTCP\_Time} \text{ MOD } (\text{SendClockFactor} \times 31,25 \mu\text{s}) \quad (15)$$

where

CycleCounterOffset	is the offset of the cycle counter
PTCP_Time	is the PTCP 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.

NOTE 2 MOD does an integer division and delivers the remainder.

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 139.

Table 139 – 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 140 and Table 141.

Table 140 – 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 141 – 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 142.

Table 142 – PTCP_MasterPriority1.Level

Value (hexadecimal)	Usage	Meaning
0x00	Level 0 (highest)	See Annex J
0x01	Level 1	
0x02 – 0x06	...	
0x07	Level 7 (lowest)	

Bit 6: PTCP_MasterPriority1.Reserved

This field shall be set zero.

Bit 7: PTCP_MasterPriority1.Active

This field shall be set zero for the PDSyncData.

This field shall be set zero for the PTCP-ANNOUNCE-PDU if the sync master doesn't convey PTCP-SYNC-PDU's and shall be set to one if the sync master conveys PTCP-SYNC-PDU's.

Thus this field shall be set to one 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 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 143.

Table 143 – 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 144. For working clock synchronization the arbitrary timescale (ARB) shall be used.

Table 144 – 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 source of time. The timescale distributed shall be ARBITRARY. A clock class 0x0D clock 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 source of time and is in holdover mode and within holdover specifications. The timescale distributed shall be ARBITRARY. A clock class 0x0E clock shall not be a slave to another clock in the PTCP domain	—
0x0F – 0xBA	Reserved	—

Value (hexadecimal)	Usage	Meaning
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 145. The PTCP_ClockAccuracy indicates the expected accuracy of a clock when it is master. The value of PTCP_ClockAccuracy shall represent its current status.

Table 145 – PTCP_ClockAccuracy

Value (hexadecimal)	Usage	Meaning
0x00 – 0x1F	Reserved	—
0x20	The time is accurate to within 25 ns	—
0x21	The time is accurate to within 100 ns	—
0x22	The time is accurate to within 250 ns	—
0x23	The time is accurate to within 1 µs	—
0x24	The time is accurate to within 2,5 µs	—
0x25	The time is accurate to within 10 µs	—
0x26	The time is accurate to within 25 µs	—
0x27	The time is accurate to within 100 µs	—
0x28	The time is accurate to within 250 µs	—
0x29	The time is accurate to within 1 ms	—
0x30 – 0xFD	Reserved	—
0xFE	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 146. 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 146 – 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 147.

Table 147 – PTCP_T2PortRxDelay

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Port receives 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 148.

Table 148 – PTCP_T3PortTxDelay

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Port transmits 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 802.

It 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 149.

Table 149 – 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 802.3 frame as shown in Figure 36.

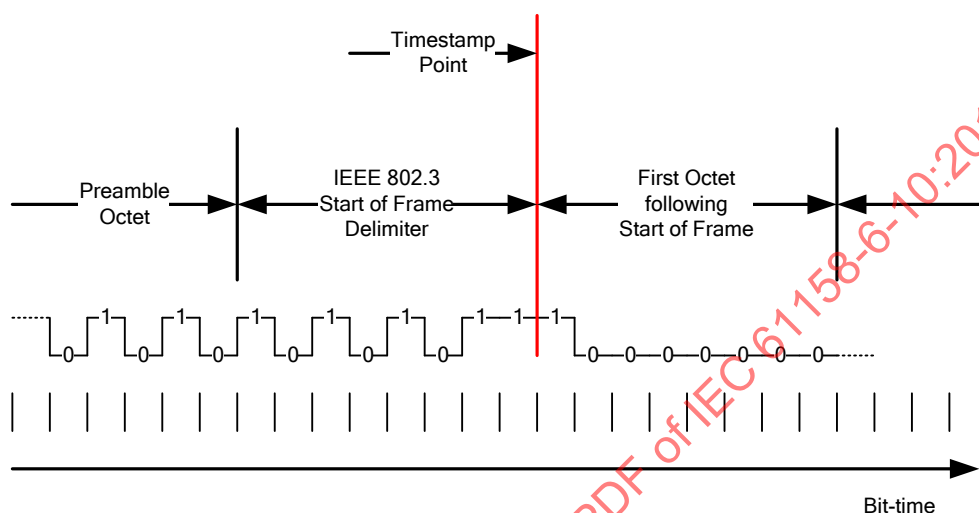


Figure 36 – Message timestamp point

4.4.4.1.2 Timer model

PTCP uses the timer model shown in Figure 37.

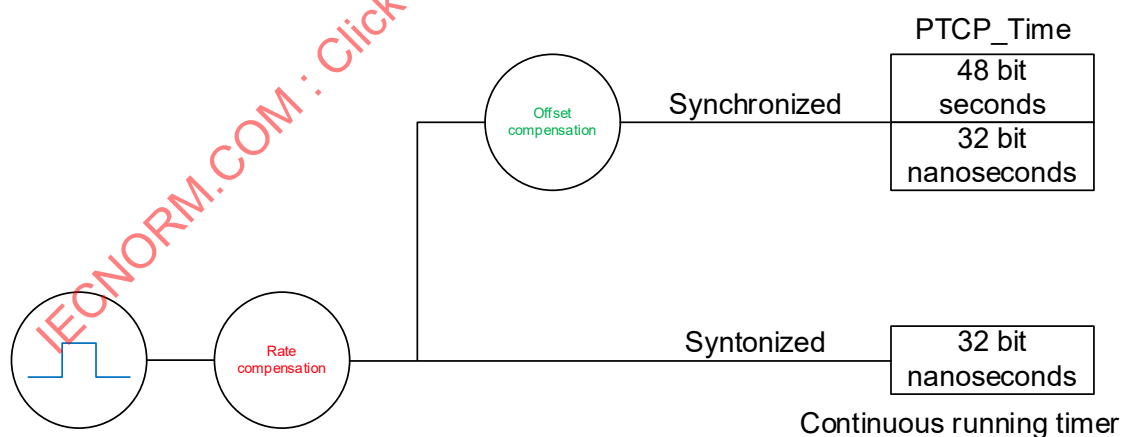


Figure 37 – Timer model

It 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 38 shows the four timestamps used for synchronization.

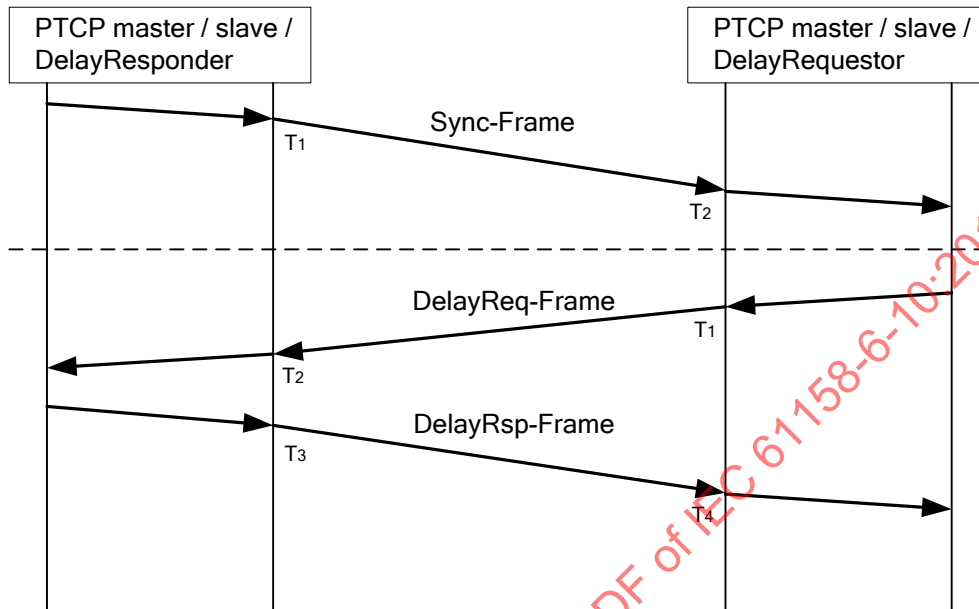


Figure 38 – Four message timestamps

The generic model uses the following formulae.

$$\text{SendTimeStamp} = T_1 \quad (16)$$

$$\text{ReceiptTimeStamp} = T_2 \quad (17)$$

$$\text{SendTimeStamp} = T_3 \quad (18)$$

$$\text{ReceiptTimeStamp} = T_4 \quad (19)$$

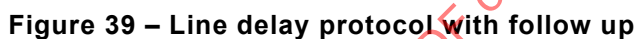
where

- 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 39. 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.

DelayResponder which 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.



The line delay measurement without DelayFuRes-message between DelayRequestor and DelayResponder is described in Figure 40.



The calculation of line delay and peer rate compensation factor is shown in following formulae. If the DelayRes-PDU contains the field T_2 , the Formula (22) shall be used, if the ResDelay is zero, RCF_{peer} shall be set to one.

$$\text{ResDelay} = T_3 - T_2 \quad (20)$$

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

$$\text{ReqDelay} = T'_4 - T'_1 \quad (21)$$

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

$$\text{RCF}_{\text{peer}} = \frac{T_2(\text{Seq}) - T_2(\text{Seq} - 1)}{T'_1(\text{Seq}) - T'_1(\text{Seq} - 1)} \quad (22)$$

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

$$\text{ResDelay}_{\text{peer}} = \frac{\text{ResDelay}}{\text{RCF}_{\text{peer}}} \quad (23)$$

where

$\text{ResDelay}_{\text{peer}}$	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 41.

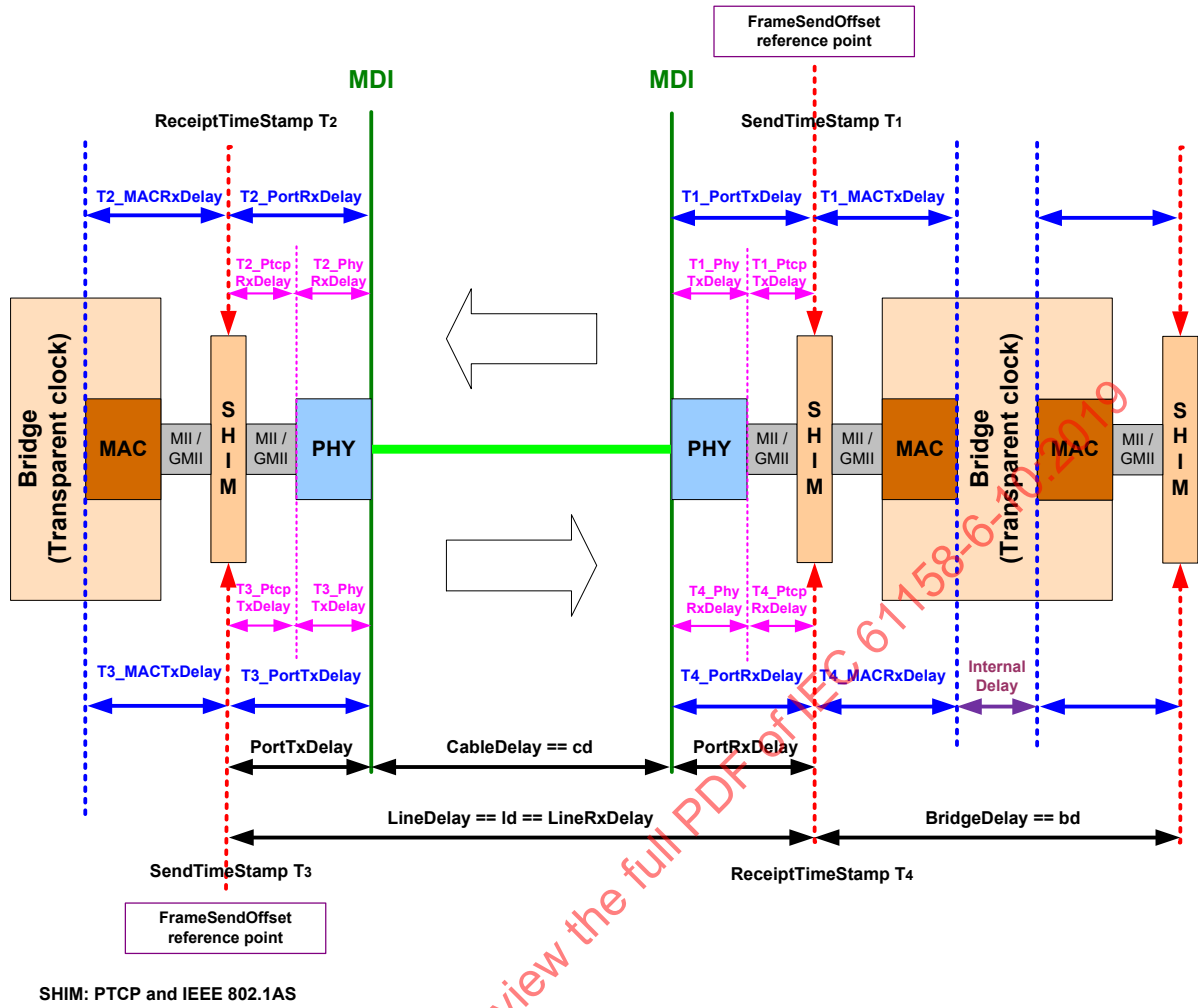


Figure 41 – Line delay measurement

NOTE 1 The Reduced Media Independent Interface (RMII) synchronizes its internal 50 MHz clock with the 25 MHz clock of the PHY. It results in a quantization error of 20 ns. This jitter cannot be compensated. The Media Independent Interface (MII) is using the PHY clock without this error.

NOTE 2 The cable type Class D has a maximal delay skew of 45 ns for 100 m cable length. It 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.

$$T1\_PortTxDelay = T1\_PhyTxDelay + T1\_PtcpTxDelay \quad (24)$$

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

$$T3\_PortTxDelay = T3\_PhyTxDelay + T3\_PtcpTxDelay \quad (25)$$

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

$$T2\_PortRxDelay = T2\_PhyRxDelay + T2\_PtcpRxDelay \quad (26)$$

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

$$T4\_PortRxDelay = T4\_PhyRxDelay + T4\_PtcpRxDelay \quad (27)$$

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

$$CableDelay = (ReqDelay - ResDelay_{peer} - T1\_PortTxDelay - T2\_PortRxDelay - T3\_PortTxDelay - T4\_PortRxDelay) / 2 \quad (28)$$

where

$CableDelay$ is the delay of the cable
 $ReqDelay$ is the delay requestor
 $ResDelay_{peer}$ is the delay responder
 $T1\_PortTxDelay$ 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

$$LineDelay = CableDelay + T3\_PortTxDelay + T4\_PortRxDelay \quad (29)$$

where

$LineDelay$ is the delay between both PTCPSHIMs for received frames
 $CableDelay$ is the delay of the cable
 $T3\_PortTxDelay$ is the port receive/transmit delay
 $T4\_PortRxDelay$ is the port receive/transmit delay

$$LineDelay_{SyncMaster} = LineDelay \times RCF_{SyncMaster} \quad (30)$$

where

$LineDelay_{SyncMaster}$ is the compensated time needed to forward a frame
 $LineDelay$ is the measured value between two peers
 $RCF_{SyncMaster}$ is the factor between the frequency of the SyncMaster and the frequency of the SyncSlave

NOTE 3 Formula (30) 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 measureable, 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) μ s, the delay between two nodes connected with 100 m Twisted Pair cabling (100 Base TX) is about 500 ns.

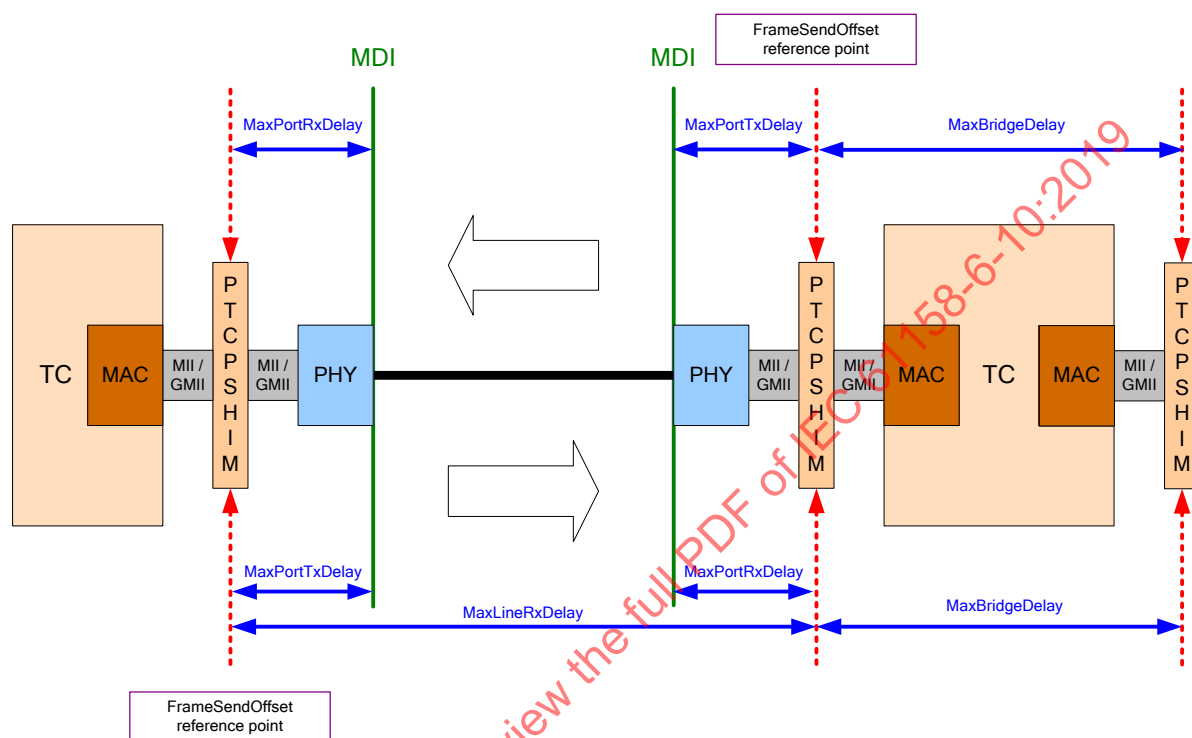


Figure 42 – Model parameter for GSDML usage

Figure 42 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 like 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 43 and adds this time to the delay field in the sync or follow up message.

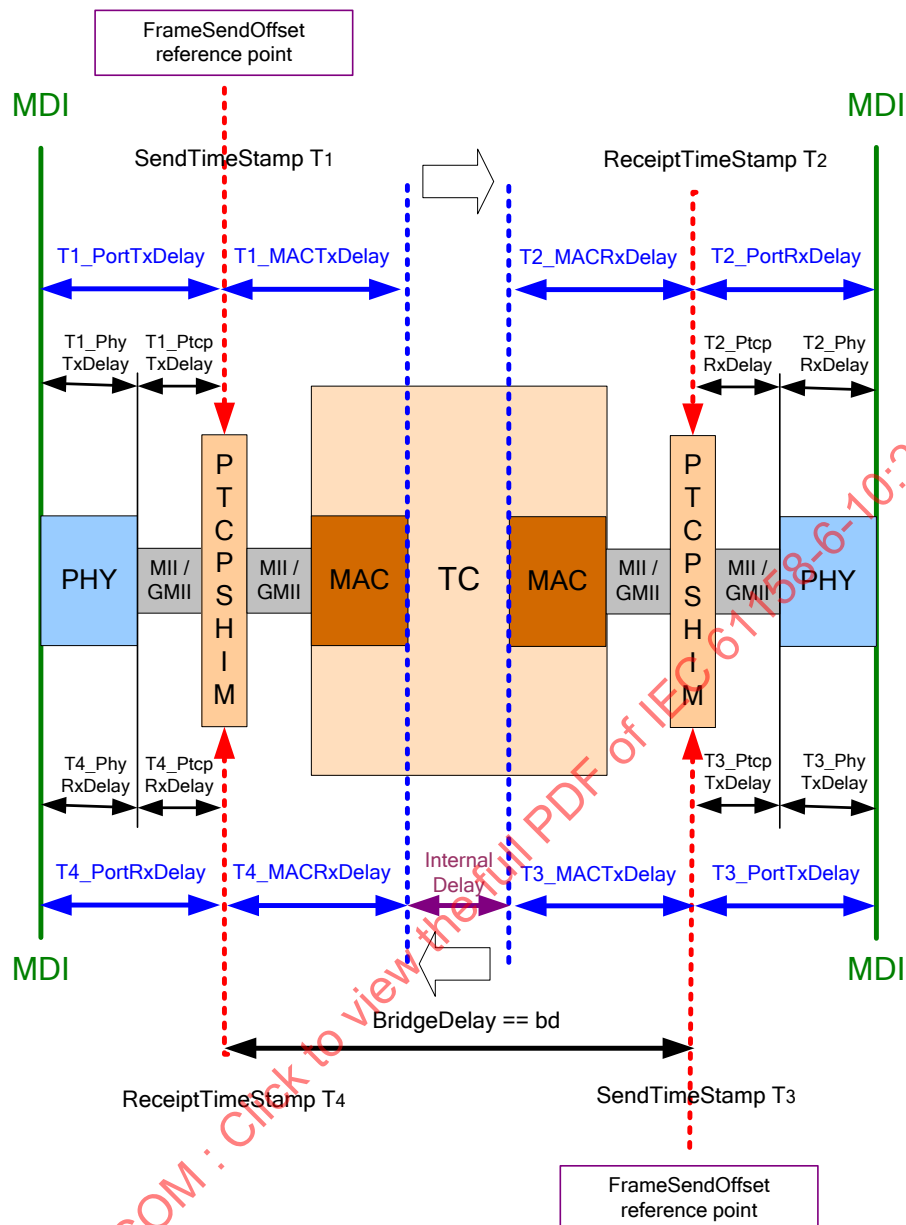


Figure 43 – Bridge delay measurement

The calculation of the bridge delay is shown in following formulae.

$$\text{BridgeDelay} = T_3 - T_4 \quad (31)$$

where

BridgeDelay	is the time needed to forward a frame
T_3	is the point in time when the first bit of the SD of a frame is transmitted
T_4	is the point in time when the first bit of the SD of a frame is received

$$\text{BridgeDelay}_{\text{SyncMaster}} = \text{BridgeDelay} \times \text{RCF}_{\text{SyncMaster}} \quad (32)$$

where

$\text{BridgeDelay}_{\text{SyncMaster}}$	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 (32) is only informative if the LineDelay is measured with the syntonized timer.

$$RCF_{SyncMaster} = \Delta t_{SyncMaster} / \Delta t_{SyncSlave} \quad (33)$$

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 through a sequence of devices can be seen in Figure 44. 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 slave. A PTCP master uses a reference to a local time system or a global time source.

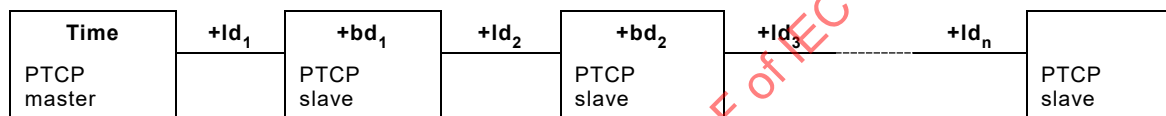


Figure 44 – Delay accumulation

The Sync-Frames are forwarded as peer to peer frames. Every device shall adjust the Sync-Frame by

- adding its bridge delay bd_x (bridge delay measurement is shown in Figure 43)
- adding the line delay ld_x (line delay measurement is shown in Figure 41),

to the delay field of the Sync-Frame as it passes through. Each PTCP slave knows the propagation delay of the sync frame. When the time of arrival T_2 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

To achieve the measurement of the deviation, see Figure 45, hardware signaling is necessary. For every supported PTCP_SyncID a signal shall be generated. It may only be accessible in test lab environment.

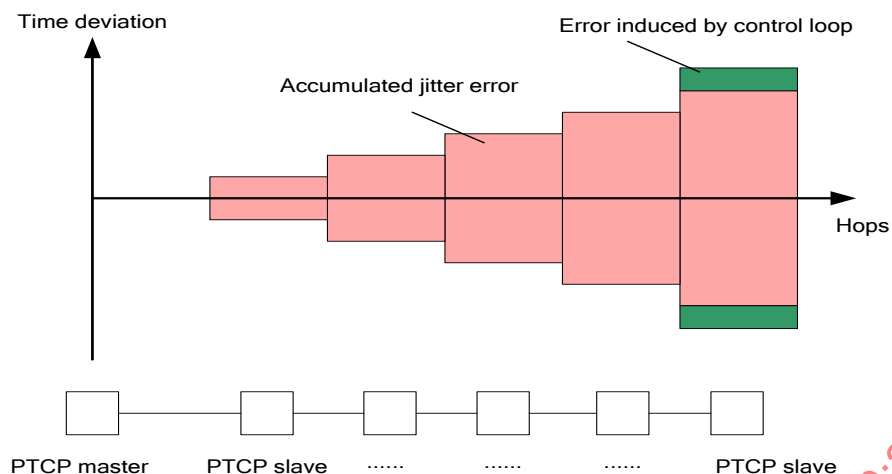


Figure 45 – Worst case accumulated time deviation of synchronization

Figure 46 and Figure 47 show in principle the measurement of the deviation for both, working clock and global time.

Every cycle (SendClock) as shown in Figure 35 shall be signaled for working clock synchronization and every second shall be signaled for global time synchronization.

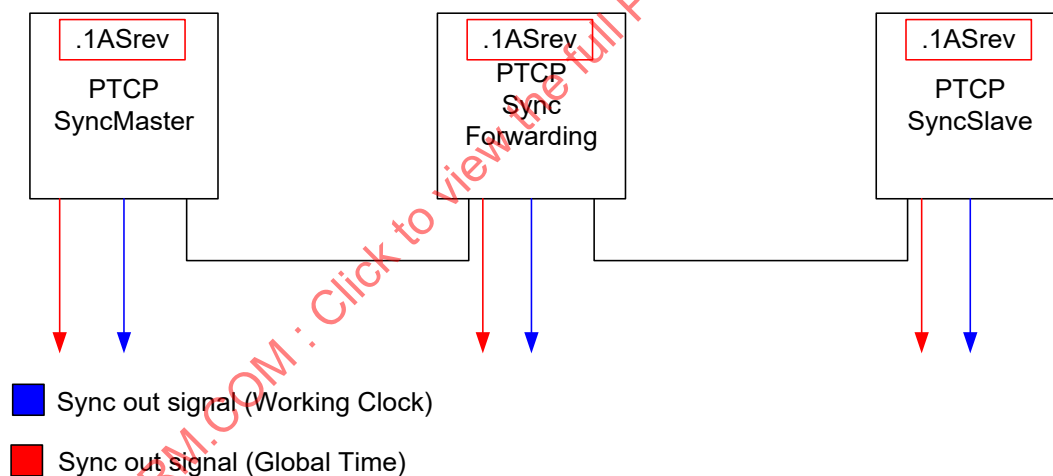


Figure 46 – Signal generation for measurement of deviation

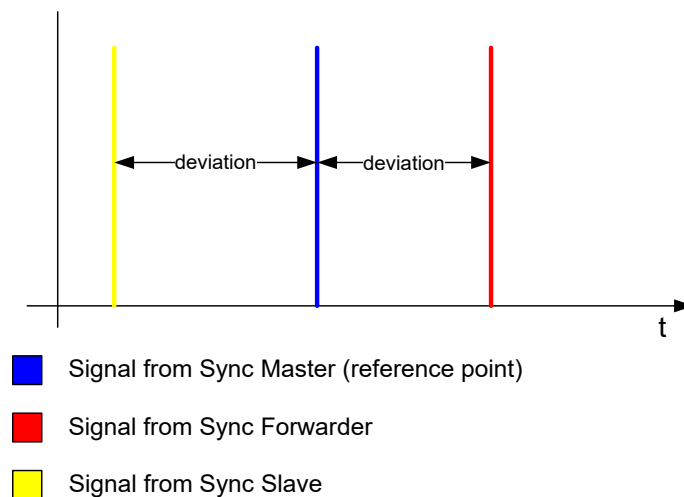


Figure 47 – Measurement of deviation

4.4.4.1.9 Synchronization Protocol

4.4.4.1.9.1 General

PTCP masters are sending Sync-Frames periodically. The sync frame has an original sending 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; i.e. the time which passed between T_{ORG} and sending 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}); i.e. 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 time stamp point; see Figure 36 for details.

Each PTCP slave compares its local clock with the sum of PTCPTime, PTCP_Delay out of the sync frame and (if applicable) PTCP_Delay out 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 are sending Sync-Frames periodically. The synchronization scheme as described in Figure 48 requires transferring the exact sending time of the Sync-Frame as initial value of the delay.

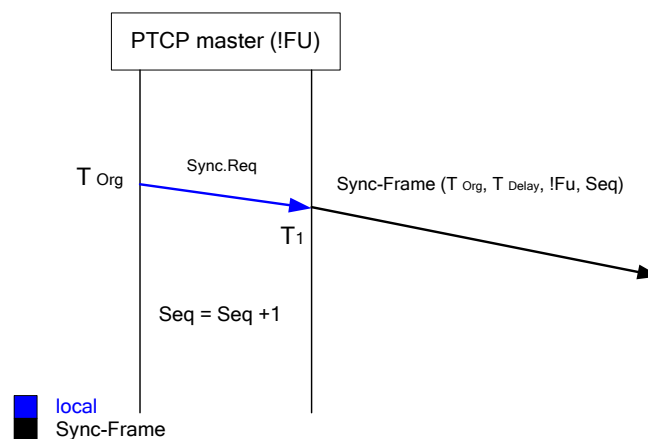


Figure 48 – PTCP master sending Sync-Frame without Follow Up-Frame

$$T_{Delay} = T_1 - T_{ORG} \quad (34)$$

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 send by the PTCP Master
T_1	is the time stamp of sending 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 49 shows a PTCP master which cannot modify a frame before transmission. A PTCP master who is not able to enter the internal handling time into the Sync-Frame directly will use a FollowUp-Frame.

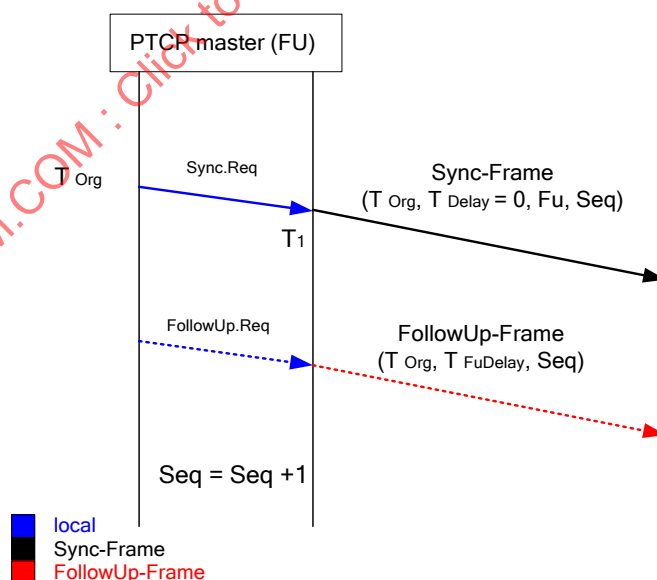


Figure 49 – PTCP master sending Sync-Frame with FollowUp-Frame

The delay field in the Sync-Frame shall be set to 0 by the PTCP master. The delay ($T_{FuDelay}$) in the associated FollowUp-Frame contains the initial value of the sync frame's delay.

$$T_{FuDelay} = T_1 - T_{Org} \quad (35)$$

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 send by the PTCP Master.
T_{Org}	is the original time of the PTCP master clock
T_1	is the time stamp of sending the Sync-Frame by the PTCP master

4.4.4.1.9.4 PTCP Slave Synchronization (!FU)

A PTCP slave as shown in Figure 50 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.

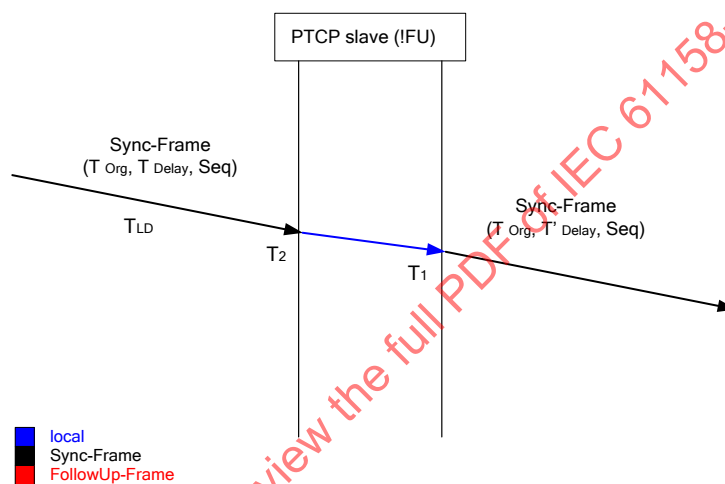


Figure 50 – !FU Sync Slave Forwarding Sync-Frame

$$T_{Delay} = T_{Delay} + T_{LD} + (T_1 - T_2) \quad (36)$$

where

T'_{Delay}	is the value of the PTCP_Delay field within the sync frame send by the PTCP slave
T_{Delay}	is the delay time as being 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 802.3 do not allow modifying a frame during transmission.

Such as PTCP slave (FU-Node) still 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.

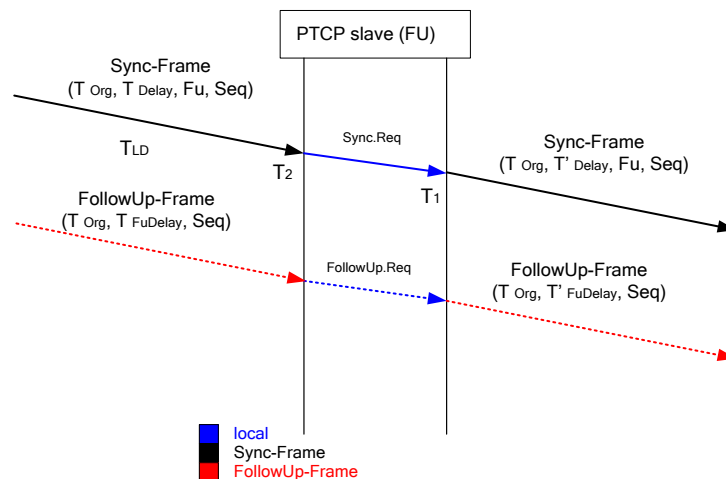


Figure 51 – FU Sync Slave Forwarding Sync- and FollowUp-Frame

$$T'_{\text{Delay}} = T_{\text{Delay}} + T_{\text{LD}} \quad (37)$$

where

T'_{Delay} is the value of the PTCP_Delay field within the sync frame send by the PTCP slave
 T_{Delay} is the delay time within the sync frame as being received by the PTCP slave
 T_{LD} is the line delay of the receiving port

$$T'_{\text{FuDelay}} = T_{\text{FuDelay}} + (T_1 - T_2) \quad (38)$$

where

T'_{FuDelay} is the value of the PTCP_Delay field within the FollowUp frame send by the PTCP slave
 T_{FuDelay} is the delay time within the FollowUp frame as being 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 is generated by an FU PTCP slave. This principle is shown in Figure 52.

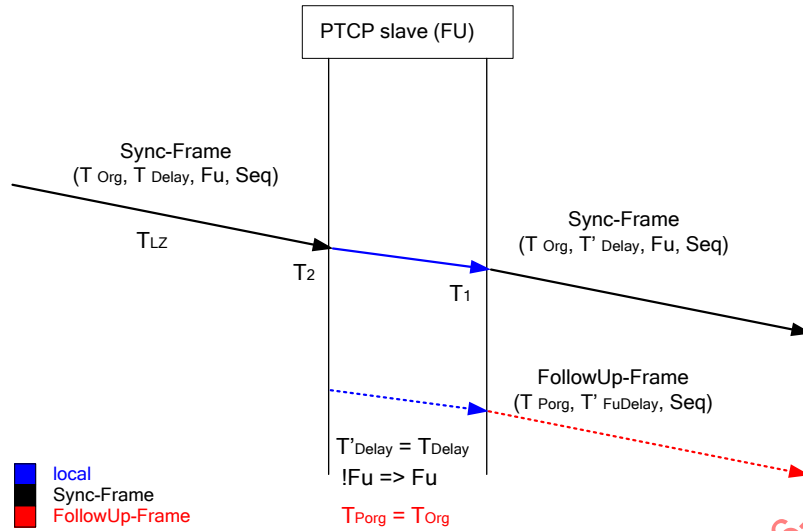


Figure 52 – FU Sync Slave Forwarding Sync- and Generating FollowUp-Frame

$$T'_{\text{Delay}} = T_{\text{Delay}} + T_{\text{LD}} \quad (39)$$

where

T'_{Delay}	is the value of the PTCP_Delay field within the sync frame send by the PTCP slave
T_{Delay}	is the delay time within the sync frame as being received by the PTCP slave
T_{LD}	is the line delay of the receiving port

$$T'_{\text{FuDelay}} = (T_1 - T_2) \quad (40)$$

where

T'_{FuDelay}	is the value of the PTCP_Delay field within the FollowUp frame send 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.

To deal with redundancy switchover in case of redundant paths, a Sync-Frame shall be forward only if the sequence number difference to the previous forwarded message is positive. This prevents duplicates which may corrupt the follow up sequence.

The use of an alternate 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 higher as the timeout of the time master and is calculated independently at every update of the line delay.

For resource limitations 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 have delay values below 1 μ s). 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 53 shows that the line delay is monitored by the LineT and 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 is increased.

If the current measurement cycle (5 times) is finished a break for DelayT (long) is made.

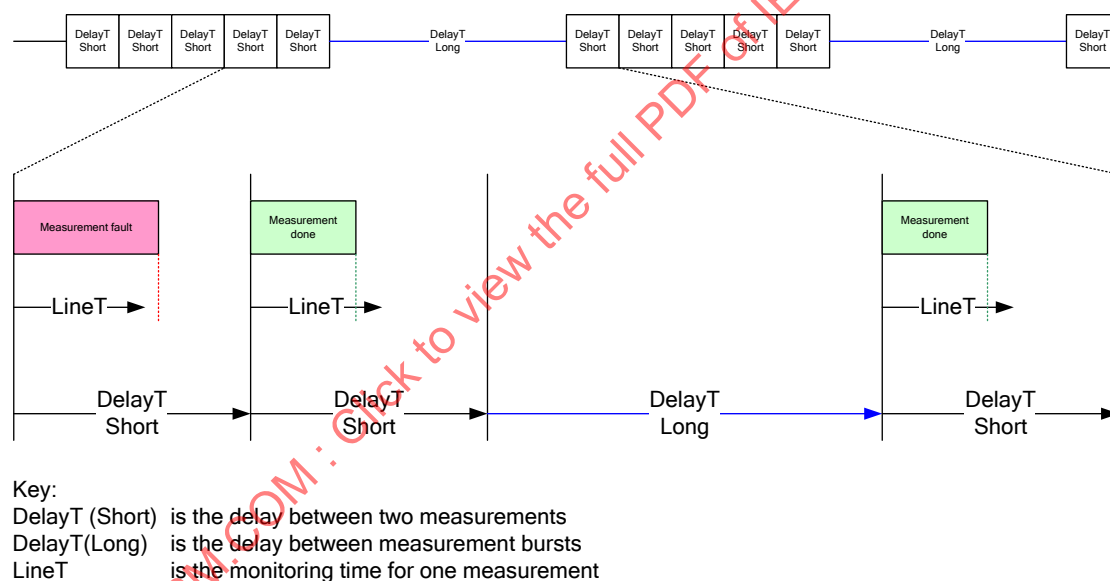


Figure 53 – Principle of the monitoring of the line delay measurement

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 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 150.

Table 150 – Remote primitives issued or received by DELAY_REQ

Primitive	Source	Destination	Associated parameters	Functions
MAUType_Change_ind	IEEE802.3	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 151.

Table 151 – 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 54.

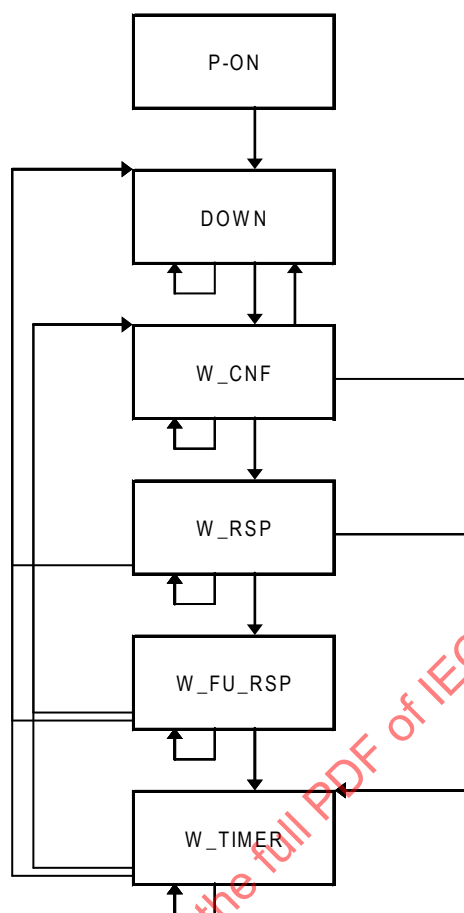


Figure 54 – State transition diagram of DELAY_REQ

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 152 contains the state table used by DELAY_REQ.

Table 152 – 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 Parameters

4.4.4.2.1.6 Functions, Macros, Timers and Variables

Table 153 contains the functions, macros, timers and variables used by the DELAY_REQ and their arguments and their descriptions.

Table 153 – 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 doesn't support line delay measurement but wants to receive the sync messages. This case is signaled by PTCP_DELAY_RSP_PDU.DelayIns:= 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 one
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.
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 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
DelayReq_Req (Port, PTCP_DELAY_REQ_PDU)	Macro	Create PTCP-PDU according 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)

Name	Type	Function/Meaning
DelayResp_Ind (S_Port, Tstamp, PTCP_DELAY_RSP_PDU)	Macro	Receive PTCP-PDU according 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 control the start of a line delay measurement
LineT	Timer	This local timer is used to monitor the timely correct 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 measurement. 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 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 154.

Table 154 – Remote primitives issued or received by DELAY_RSP

Primitive	Source	Destination	Associated parameters	Functions
MAUType_Change_ind	IEEE802.3	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 155.

Table 155 – 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 55.

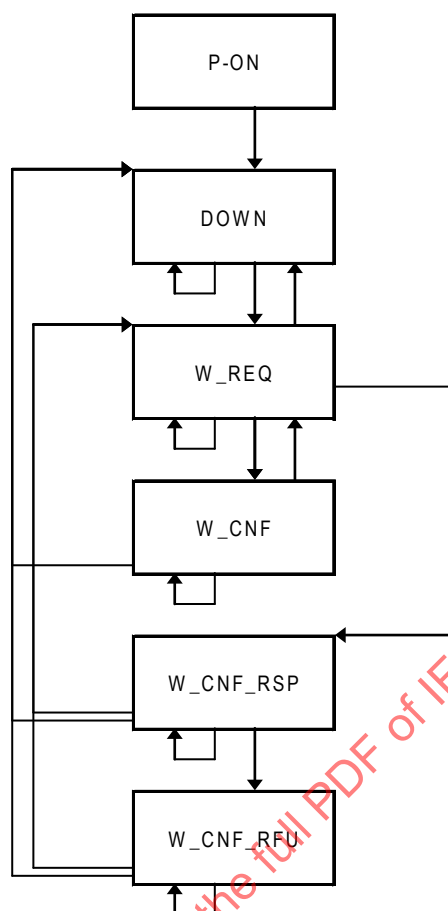


Figure 55 – State transition diagram of DELAY_RSP

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 with a delay response message.

4.4.4.2.2.4 DELAY_RSP state table

Table 156 contains the state table used by DELAY_RSP.

Table 156 – 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 157 contains the functions, macros, timers and variables used by the DELAY_RSP and their arguments and their descriptions.

Table 157 – 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 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 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 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 time stamp of the delay request message.
Tx_Tstamp_T3	Variable	This local variable contains the transmit time stamp 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 for master slave state tables

Figure 56 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.

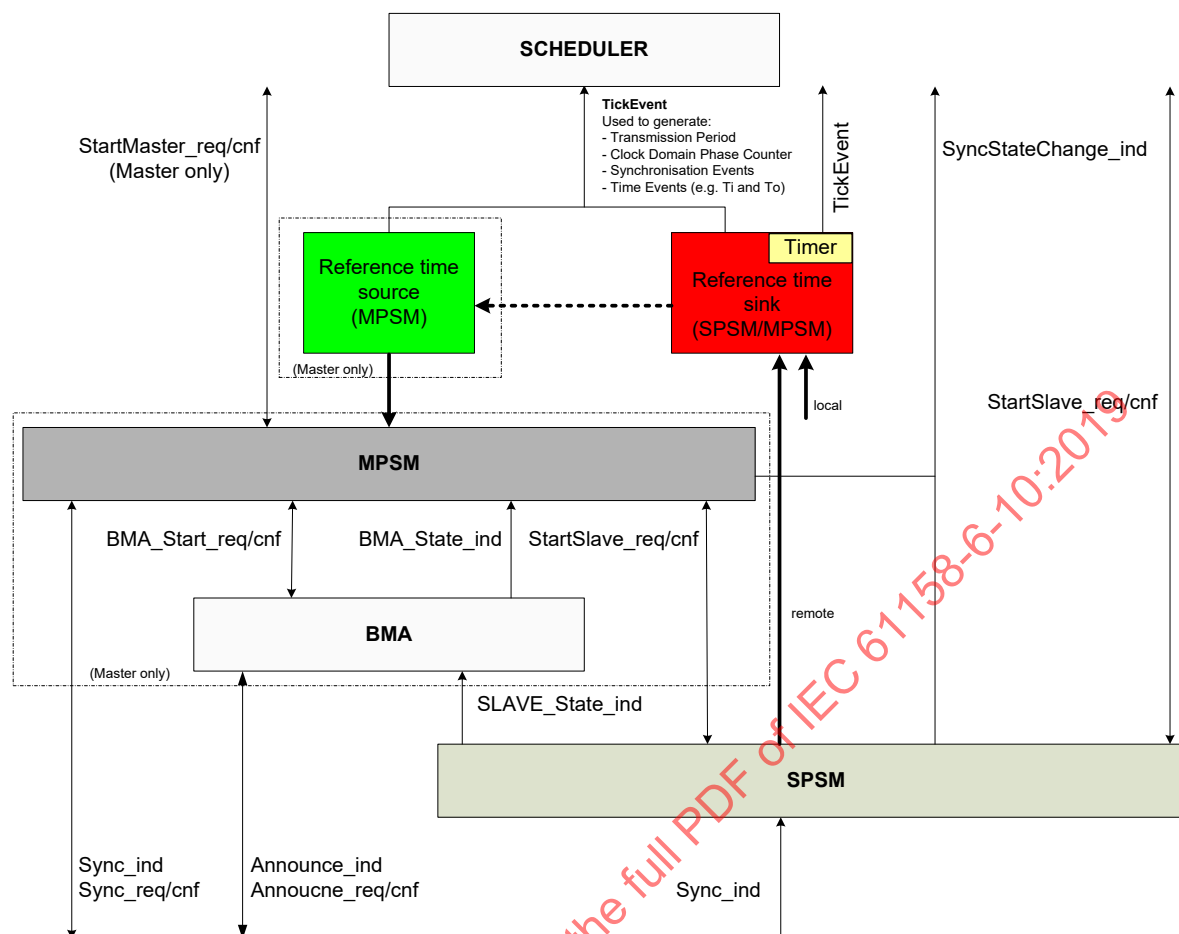


Figure 56 – Overview of PTCP

For the use of a PTCP in conjunction with wireless the principles of IEEE 802.1AS should be used.

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 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 158.

Table 158 – 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 159.

Table 159 – 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	—	Deactivates 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 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 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 57:

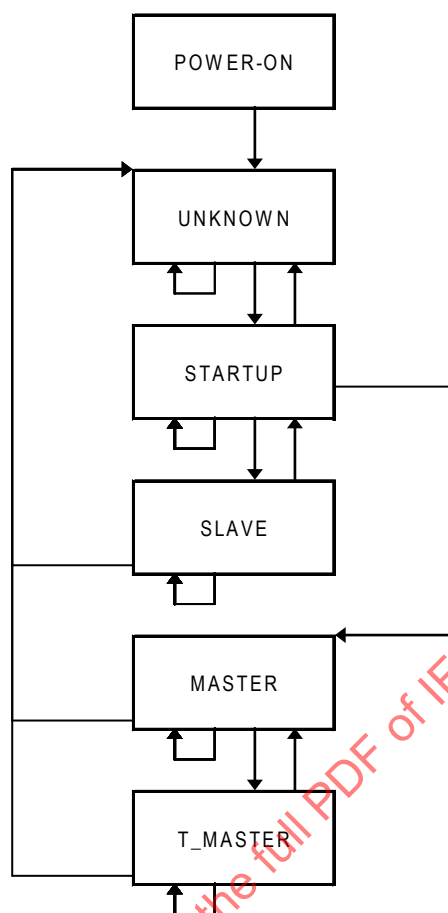


Figure 57 – State transition diagram of SYN_BMA

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 as the best master.
T_MASTER	This is a transition state which indicates the receiving 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 local 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 his 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 master election will be made in two steps. With the first received announce the potentially master will be stored in a master list. With the next announce the SYN_BMA elects this master, if it's the best known master.

During master competition all potentially masters will be collected in master list. After election they drop out via altering.

If the decision to be master is dropped, the best master starts sending sync messages. The remaining potential masters stop to send announce messages and have to be slaves. If the active master fails, the backup masters restart sending sync and announce messages.

4.4.4.4.1.4 SYN_BMA state table

Table 160 contains the state table used by SYN_BMA.

Table 160 – 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 161 contains the functions, macros, timers and variables used by the SYN_BMA and their arguments and their descriptions.

Table 161 – 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 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 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	Executes comparison according 4.4.4.1.6 and return TRUE if the local master is found, else FALSE.
TRANSMIT_PORT_VALID	Macro	Table 176 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 of 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 found the “to be used” sync master is done by the BMA using the keys 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) Checking 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 it continuous 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 of 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 of 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 of 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 of 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 octet zero to five) of the PTCP_MasterSourceAddress of the residual list.

Use the found one.

NOTE 4 The PTCP_MasterSourceAddress is used as “tie breaker”.

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 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 162.

Table 162 – 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 163.

Table 163 – 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 for master start
StopMaster_req	PTCP ASE	MPSM	—	Deactivates master function
StopMaster_cnf	MPSM	PTCP ASE	Status	Confirmation for master stop
StartSlave_req	MPSM	SPSM	—	Activate slave function
StartSlave_cnf	SPSM	MPSM	—	Confirmation for start slave
StopSlave_req	MPSM	SPSM	—	Deactivates slave function
StopSlave_cnf	SPSM	MPSM	—	Confirmation for stop slave
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	—	Deactivates 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
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 58:

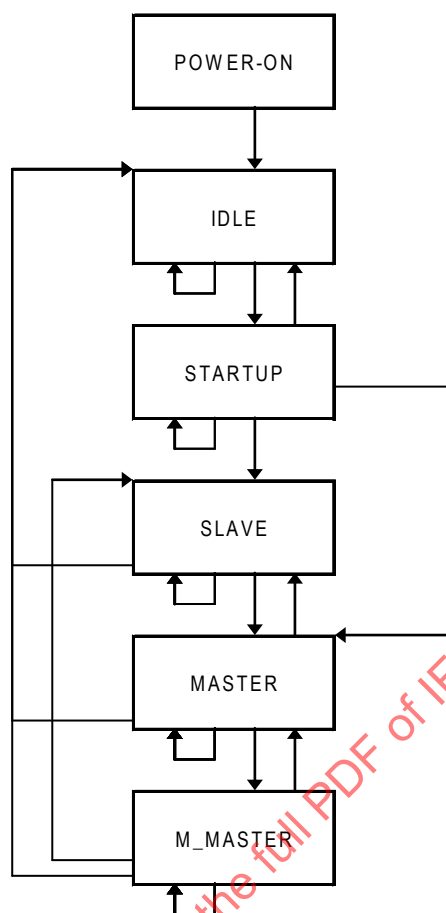


Figure 58 – State transition diagram of SYN_MPSM

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 goes 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 made local by every master for every supported PTCP SyncID and the engineering assigned PTCP_SubdomainUUID. 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 grant 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 comes along, the new master needs to synchronize to the active master. For synchronization the MPSM goes to state SLAVE.

4.4.4.5.4 SYN_MPSM state table

Table 164 contains the state table used by SYN_MPSM.

Table 164 – 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

#	Current State	Event /Condition =>Action	Next State
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
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

#	Current State	Event /Condition =>Action	Next State
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
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

#	Current State	Event /Condition =>Action	Next State
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
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 165 contains the functions, macros, timers and variables used by the SYN_MPSM and their arguments and their descriptions.

Table 165 – 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 170 shall be applied.
RECEIVE_PORT_VALID	Macro	Table 175 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 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 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 176 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 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 166.

Table 166 – 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 167.

Table 167 – 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 for start slave
StopSlave_req	MPSM	SPSM	—	Deactivates slave function
StopSlave_cnf	SPSM	MPSM	Status	Confirmation for stop slave
SyncStateChange_ind	SPSM	FSPM	Status	Indication for events – JITTER_OUT_OF_BOUNDARY – NO_SYNC_MESSAGE_RECEIVED
SLAVE_State_ind	SPSM	BMA	Status	Indication for 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 59.

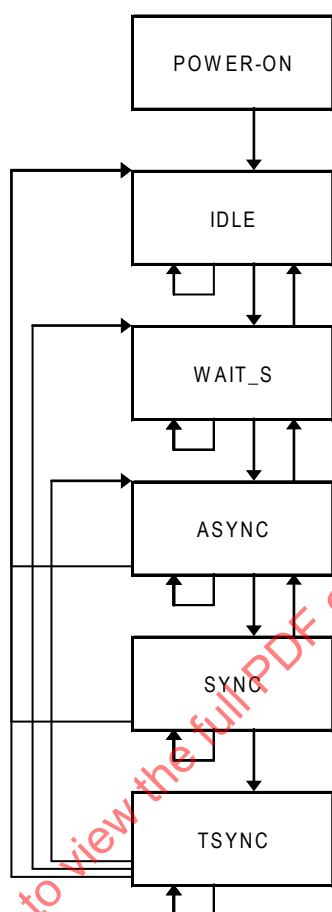


Figure 59 – State transition diagram of SYN_SPSM

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 goes 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 goes 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 will be received the SPSM shall switch to state TSYNC.

4.4.4.6.4 SYN_SPSM state table

Table 168 contains the state table used by SYN_SPSM.

Table 168 – 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"	ASYN
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)	ASYN
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 169 contains the functions, macros, timers and variables used by the SYN_SPSM and their arguments and their descriptions.

Table 169 – 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 makes the offset compensation Parameters: – LocalTime – MasterTime
OFFSET_CTRL_IS_SYNC	Macro	This local macro measured the sync error in PLL Window. It shall only return TRUE if staying 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 initialize 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 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 170 shall be applied. Additional a valid RATE shall exist.
OffsetCtrlLoopT	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 lesser 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 additional 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 170 contains the truth table used by the SPSM.

Table 170 – Truth table for one SyncID for receiving sync and follow up frames

Input								Output
Condition of the receive port ^a								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 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 Primitives exchanged between remote machines

The 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 171.

Table 171 – 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 172.

Table 172 – 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 60.

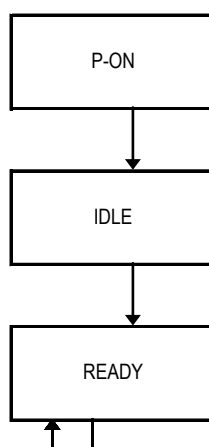


Figure 60 – State transition diagram of SYNC_RELAY

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 for both.

NOTE A SYNC_RELAY of a PTCP master never forward sync messages created by the local PTCP master.

Each time stamp unit of the bridge shall correct the delay field of sync frames on the fly. 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 circulating sync messages in the network.

Announce frames shall be handled by the MAC_RELAY. Additional 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 173 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 175 and Table 176 show at a glance the conditions.

Table 173 – 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 && !INVALID_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 174 contains the functions, macros, timers and variables used by the SYNC_RELAY and their arguments and their descriptions.

Table 174 – 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 the first received PTCP_Sync frame over a valid link. The entry is thrown out if a different PTCP_SubdomainUUID is parameterized later. Case "PTCP_SubdomainUUID is parameterized" An entry is made automatically the first received PTCP_Sync frame over 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 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 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 170 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 175 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 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 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 176 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 receive port and sequence number is valid 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 175 and Table 176 contain the truth table used by the SYNC_RELAY.

Table 175 – 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 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 of the PTCP_Delay1ns / PTCP_Delay10ns fields.

Table 176 – 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 service primitives including their associated parameters issued or received by SCHEDULER are described in the service definition and shown in Table 177.

Table 177 – 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 178.

Table 178 – 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 61.

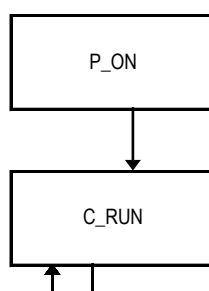


Figure 61 – State transition diagram of SCHEDULER

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 generates the signals for the local handling of time events. The TickEvent is additionally used to trigger the timely correct forwarding of RT_CLASS_3 frames.

4.4.4.8.4 SCHEDULER state table

Table 179 contains the state table used by SCHEDULER.

Table 179 – 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
10	C_RUN	TickEvent (CycleCounter, CycleOffset) /RxPeriod == GREEN && RxPeriodChecker () == RED =>	C_RUN

#	Current State	Event /Condition =>Action	Next State
		RxPeriod:= RED DMUX_Schedule_req (CycleCounter, RxPeriod)	
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 180 contains the functions, macros, timers and variables used by the SCHEDULER and their arguments and their descriptions.

Table 180 – Functions, Macros, Timers and Variables used by the SCHEDULER

Name	Type	Function/Meaning
RxPeriodChecker	Function	This local function checks the active RxPeriod and returns it.
TickEvent	Function	Local generated event from a timer. The timer shall be controlled by the PTC master or slave in case of synchronization or free running without synchronization. Parameter: – CycleCounter – CycleOffset
TxPeriodChecker	Function	This local function checks the active TxPeriod and returns it.
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)
RxPeriod	Variable	This variable contains the values: RED and GREEN
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 181 and Table 182 contain the truth table used by the SCHEDULER to detect the required period per port.

4.4.4.8.6.2 Receive port PeriodChecker

Table 181 shows the truth table for a receive port.

Table 181 – 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 182 shows the truth table for a transmit port.

Table 182 – 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 a master and its slaves. Two timescales are used for this document as shown in Table 183.

Table 183 – Timescales

Timescale	Used protocol	Meaning
GlobalTime	IEEE 802.1AS	Used to provide the global understanding of Wall Clock time for the usage in this document
WorkingClock	PTCP	An arbitrary timescale used in a given domain to align a group of devices to a common working clock.

4.5.2 GlobalTime

For time synchronization, synchronization of the local maintained GlobalTime, IEEE 802.1AS applies. The support of the IEEE 802.1AS Management Information Base (MIB) is optional. Table 183 and Table 184 show the relation between the timescales and Figure 62 the used timer model for GlobalTime.

Table 184 – Timescale correspondence between GlobalTime, TAI and UTC

GlobalTime		TAI (International Atomic Time) Value	UTC (Coordinated Universal Time) Value
Value Seconds [48 bit]	Value Nanoseconds [32 bit]		
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 ISO 8601, where 'T' signals that a time follows the date and 'Z' signals 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 containing information about the quality of the GlobalTime is maintained.

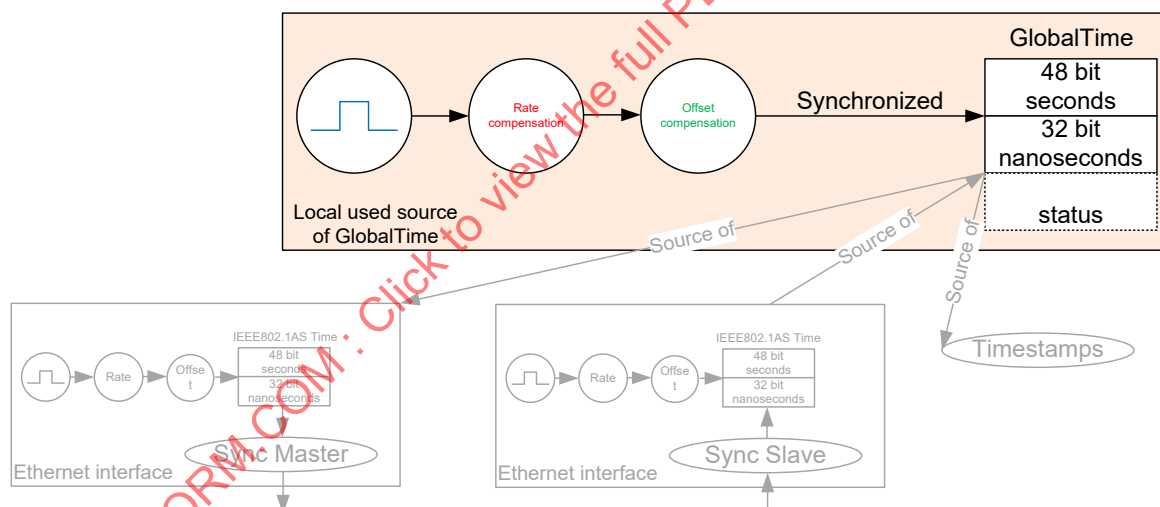


Figure 62 – GlobalTime timer model

4.5.3 WorkingClock

For working clock synchronization, synchronization of the local maintained WorkingClock, PTPC applies. Table 183 shows the used timescale and Figure 63 the used timer model for WorkingClock.

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 containing information about the quality of the WorkingClock is maintained.

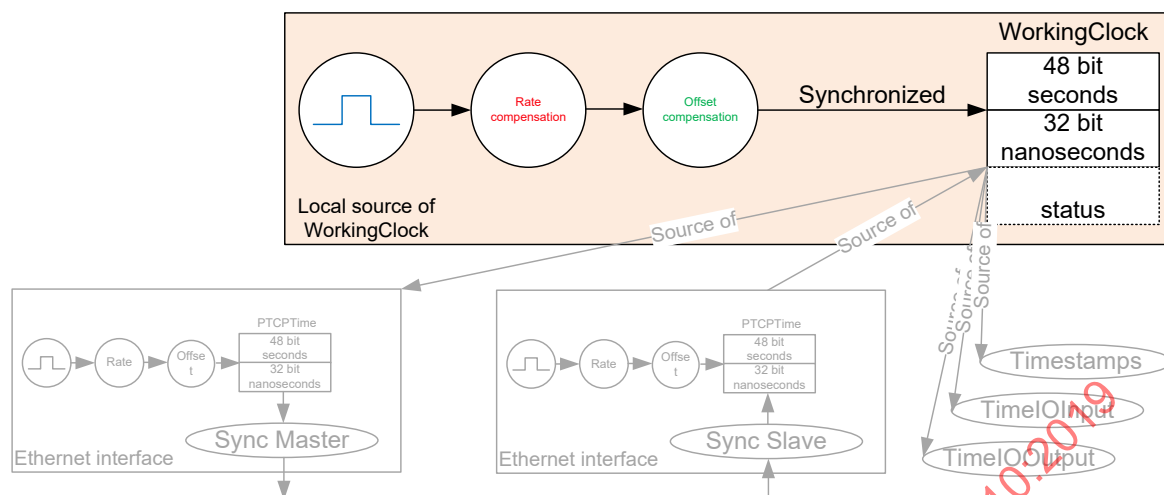


Figure 63 – WorkingClock timer model

4.6 Media redundancy

4.6.1 Media redundancy and loop prevention

4.6.1.1 General

For media redundancy the IEC 62439-2 applies, except that the support of MRP-MIB is optional. Annex W 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 64.

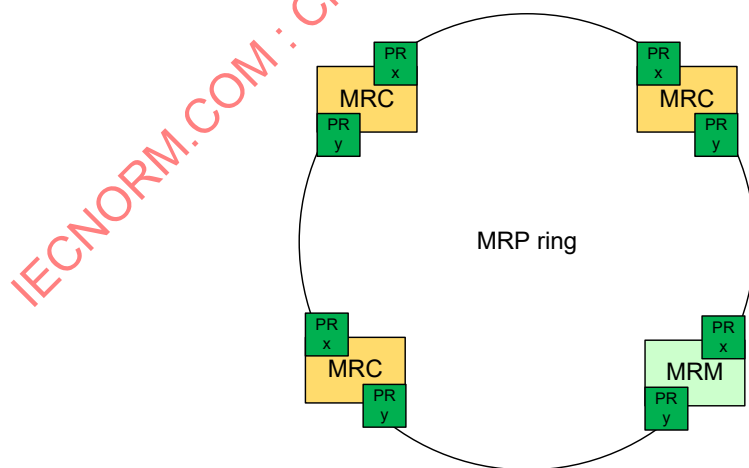


Figure 64 – Media redundancy – Ring

4.6.1.2.2 Default parameter for MRP_Role:= “Media Redundancy Client”

The “MRC parameters and constraints” parameter set from the 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 the IEC 62439-2 applies.

The MRP_Prio shall be set according Table 185.

Table 185 – 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 705 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 the IEC 62439-2 including Annex A applies.

4.6.1.2.4 Additional forwarding rules

For vendor specific applications, for example “RedApplication”, which requires in conjunction with System Redundancy redundant transport, each “Media Redundancy Client” and each “Media Redundancy Manager” shall support an additional forwarding rule according to Table 186 for the MAC address shown in Table 187.

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 186 – 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 187 – Managed Multicast MAC address

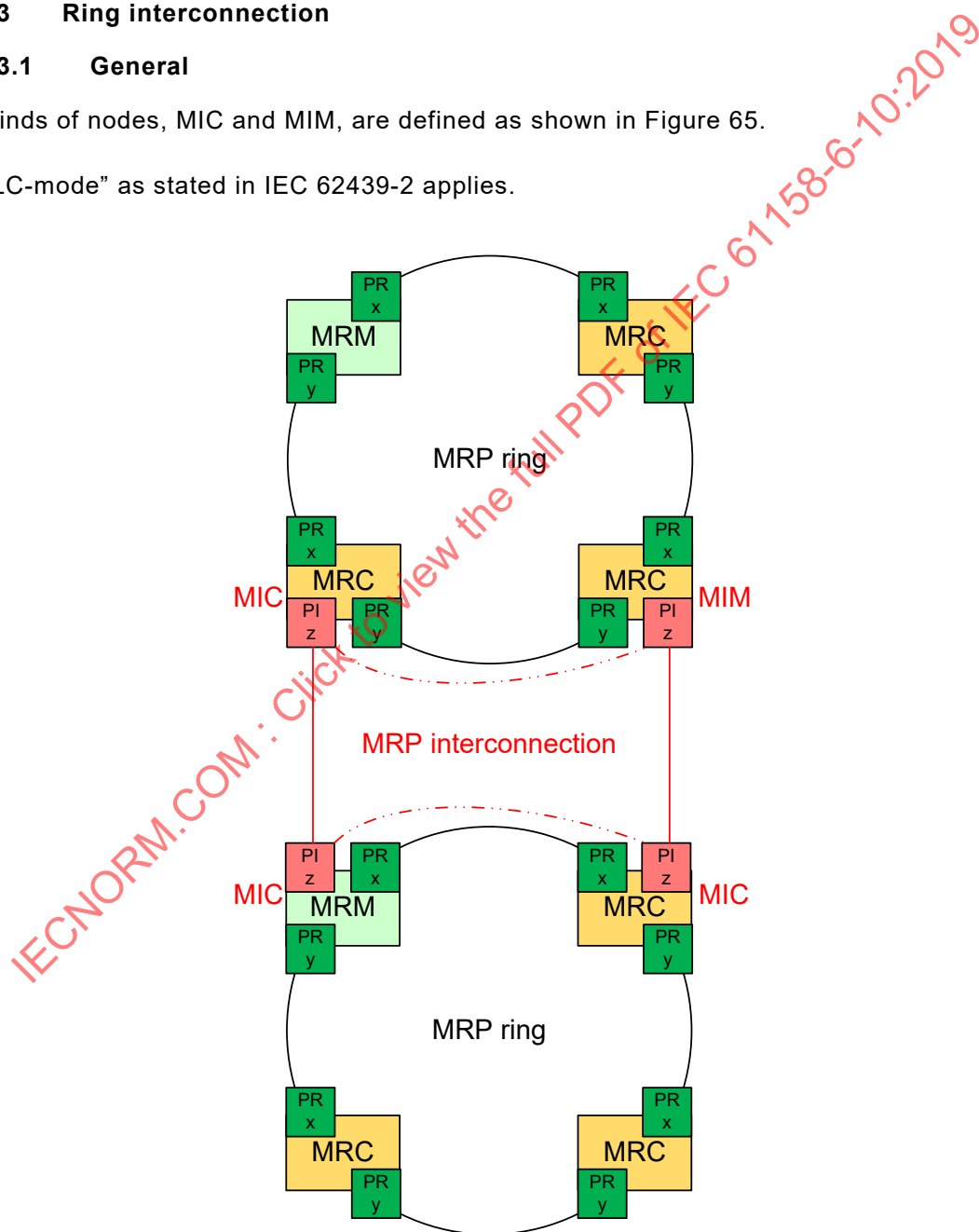
Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Comment
01-0E-CF	00-05-00	Managed Multicast MAC address for vendor specific usage in conjunction with MRP. A vendor specific EtherType shall be used.

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 65.

The “LC-mode” as stated in IEC 62439-2 applies.

**Figure 65 – 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 the 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 the 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 at the same time two ring ports.
- 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

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.

4.7 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 188 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 188 shall represent the content of the DLSDU in Table 10.

Table 188 – 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 Dataltem size. In case of UDP-RTC-PDU the number of padding octets shall be in the range of 0...12 depending on the Dataltem size.	
^b In case of RT_CLASS_3 transportation the number of padding octets is given by the engineering system.	

Table 189 defines structures for substitutions of elements of the APDU structures shown in Table 188.

Table 189 – 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 not larger than 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 field 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 67 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 66.

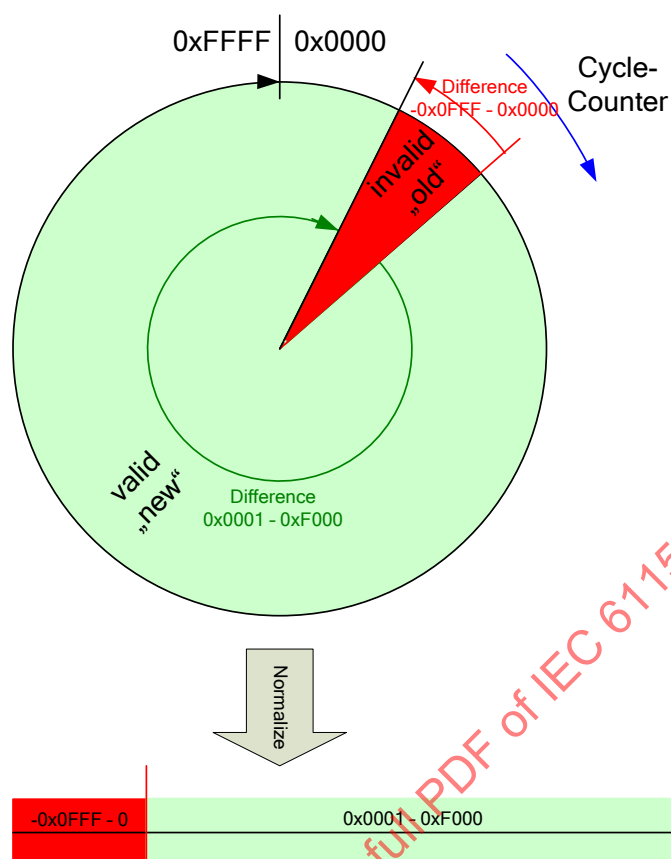


Figure 66 – CycleCounter value range

NOTE The normalization can be done using the following formula:

$\text{CycleCounter(Normalized)} := (((\text{CycleCounter(NEW)} + (0x10000 - \text{CycleCounter(STORED)})) - 1) \& 0xFFFF) - 0xF000) \times (-1)$

Table 190 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 current received PDU.

Table 190 – 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 66 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 66 15/16 of the CycleCounter range The consumer votes the received frame as newer than the stored frame	Frame is processed

If MRPD 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 μ s as shown in Figure 67.

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 identically to bit 15 of the 64 bit local counter.

The bits 5 to 19 shall be used as bit 0 to 14 identically for the CycleCounter of the UDP-RTC-PDUs. The bit 15 of the 16 bit CycleCounter of the UDP-RTC-PDUs may also be identically to bit 20 of the 64 bit local counter.

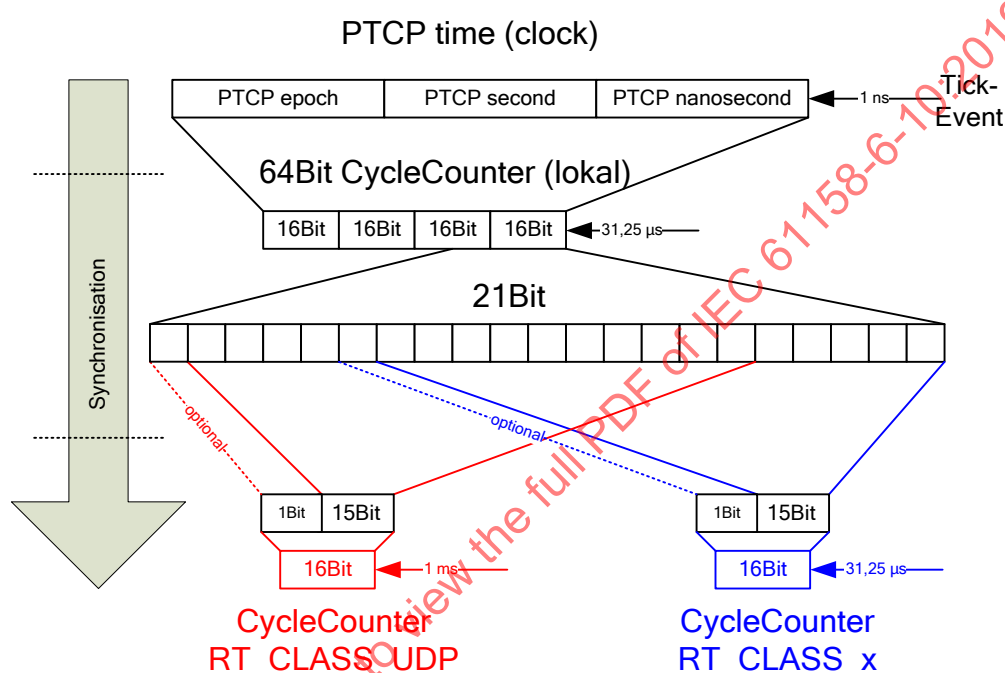


Figure 67 – Structure of the CycleCounter

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 67 and Figure 68 to avoid a false “older” detection by the receiver.

The CycleCounter shall be set due to synchronization according to the model shown in Figure 68.

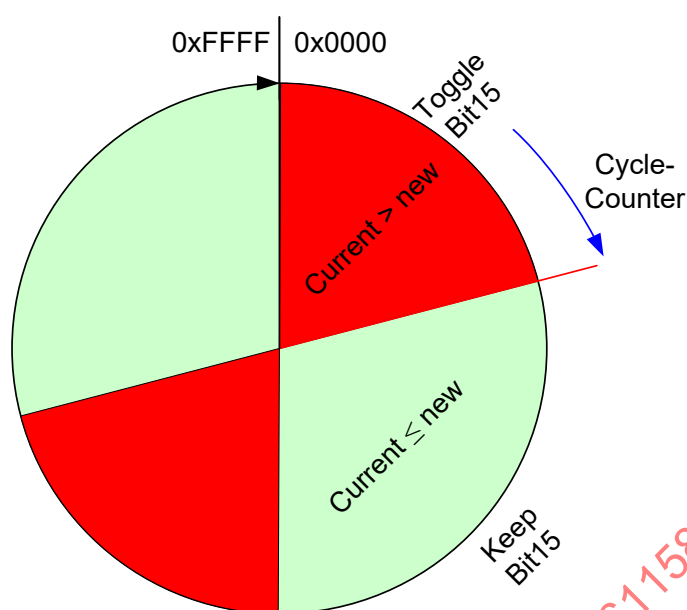


Figure 68 – Optimized CycleCounter setting

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 191 and evaluated after the AR is in state InDATA.

Table 191 – 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 192 and Table 193, and evaluated after the AR is in state InDATA.

NOTE An IO controller sets this flag independently from the ARType to the Default value.

Table 192 – DataStatus.Redundancy in conjunction with DataStatus.State==Backup

Value (hexadecimal)	Meaning
0x00	Default One primary AR of a given AR-set is present
0x01	None primary AR of a given AR-set is present

Table 193 – 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 194.

Table 194 – DataStatus.DataValid

Value (hexadecimal)	Meaning
0x00	Dataltem invalid
0x01	Dataltem valid

For a Subframe the DataStatus.DataValid shall be set to “Dataltem 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 195.

Table 195 – DataStatus.ProviderState

Value (hexadecimal)	Meaning
0x00	Stop
0x01	Run

Bit 5: DataStatus.StationProblemIndicator

This field shall be coded with the values according to Table 196.

Table 196 – 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 197 and Table 198.

Table 197 – 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 198 – 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 199.

Table 199 – 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 appeared
1	WrongLengthError	0	No length error
		1	A length error has appeared
2	MACReceiveBufferOverflow	0	No MAC receive buffer overflow
		1	A MAC receive buffer overflow has appeared
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 has not had the expected frame type or has not had the expected frame ID or has not had 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 have not arrived on time
Other	—	Reserved	Reserved

4.7.2.1.5 Coding section related to decentralized periphery

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 (41) as a 16 Bit-polynomial.

$$\text{SFCRC16} = 1 + x^2 + x^{15} + x^{16} \quad (41)$$

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 could be checked with <http://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, as shown in Figure 69 starts with the first octet of the SFPosition and ends with the last octet of the field C_SDU.

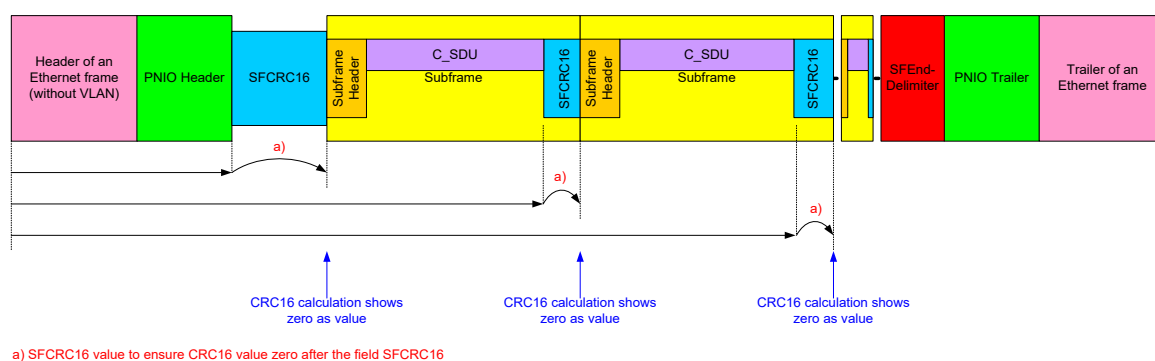


Figure 69 – SFCRC16 generation rule

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 200.

Table 200 – SFPosition.Position

Value (hexadecimal)	Meaning
0x00	End of Subframe coding
0x01 – 0x3F	Useable for packetizing and unpacking of Subframes during transport and end to end
0x40 – 0x7F	Useable for packetizing and unpacking of Subframes end to end

NOTE End to end means, that the source of data creates and concatenates the Subframes for a frame. The conveyance is done without any change on 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 201.

Table 201 – 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 202.

Table 202 – 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 70.

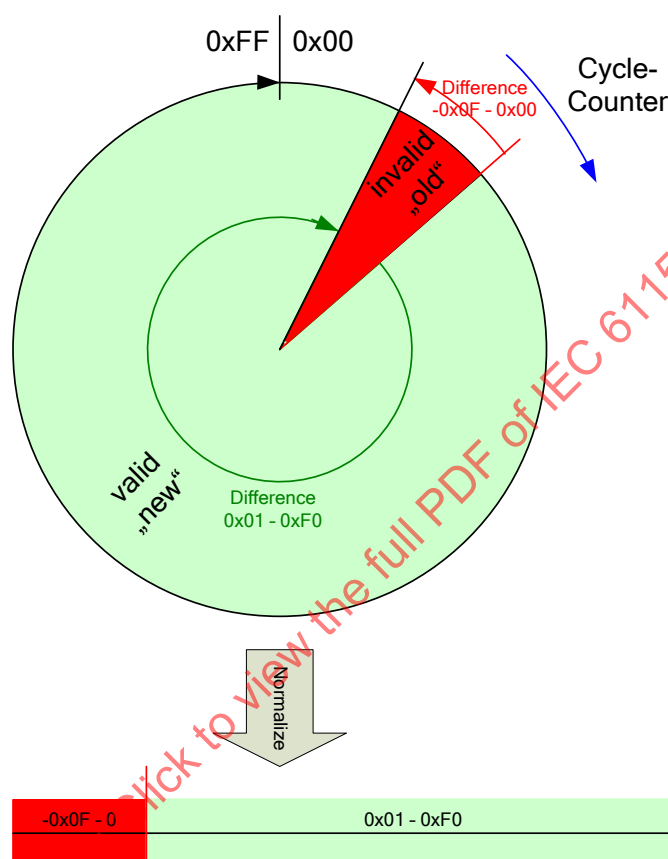


Figure 70 – SFCycleCounter value range

Table 203 defines a 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 current received Subframe.

Table 203 – SFCycleCounter Difference

Value (hexadecimal)	Time range RTC-PDU [s]	Time range UDP-RTC- PDU [s]	Meaning	Use
-0x0F– 0x00	—	—	Red sector according to Figure 70 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 70 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 204.

Table 204 – 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 205 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 205 – 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 206.

Table 206 – IOxS.DataState

Value (hexadecimal)	Meaning
0x00	Bad Data field may not contain valid 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 Provider protocol machine****4.7.4.1.1 General**

Figure 71 and Figure 72 show the basic structure of the Provider protocol machine (PPM). Table 207 shows the APDU_Status of a PPM with subframe structure.

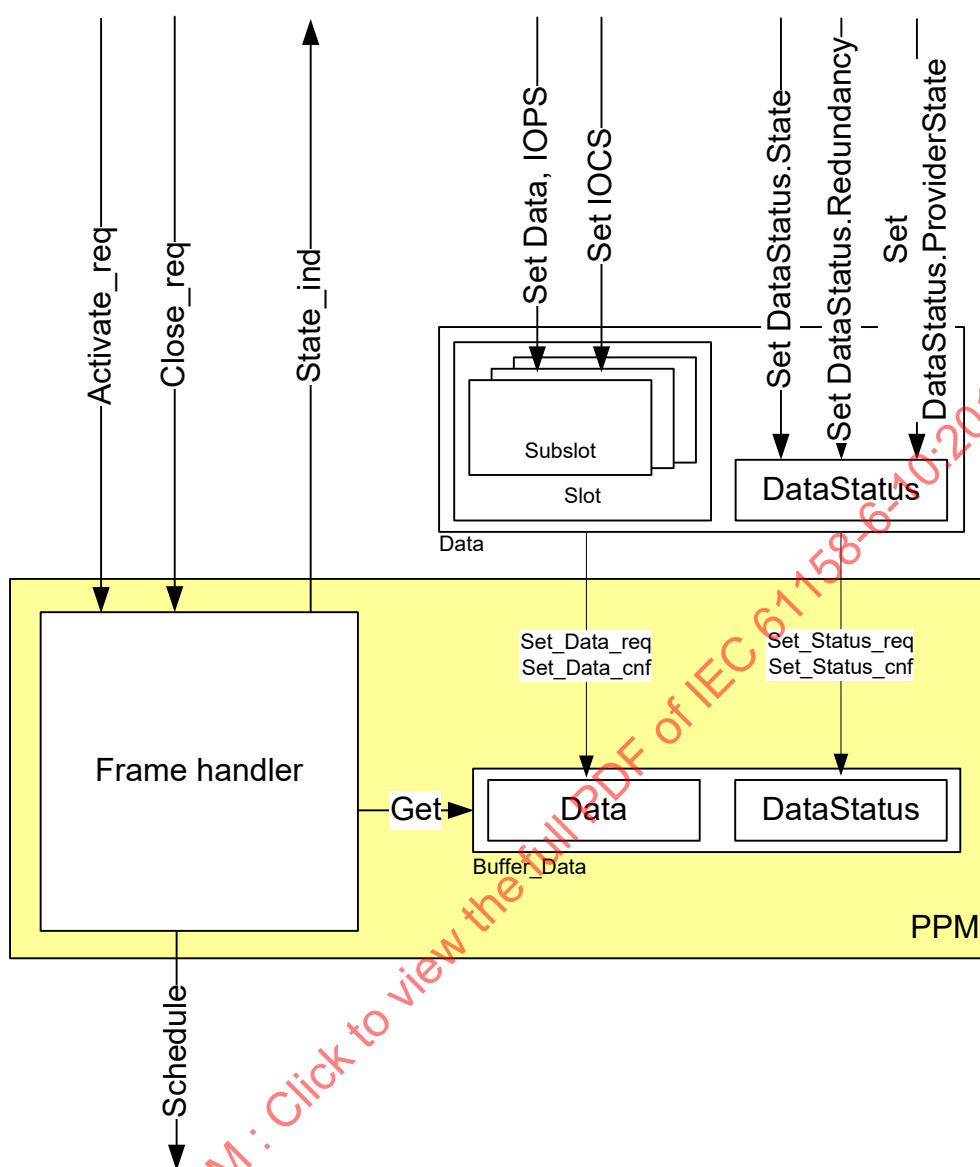


Figure 71 – Basic structure of a PPM with frame structure

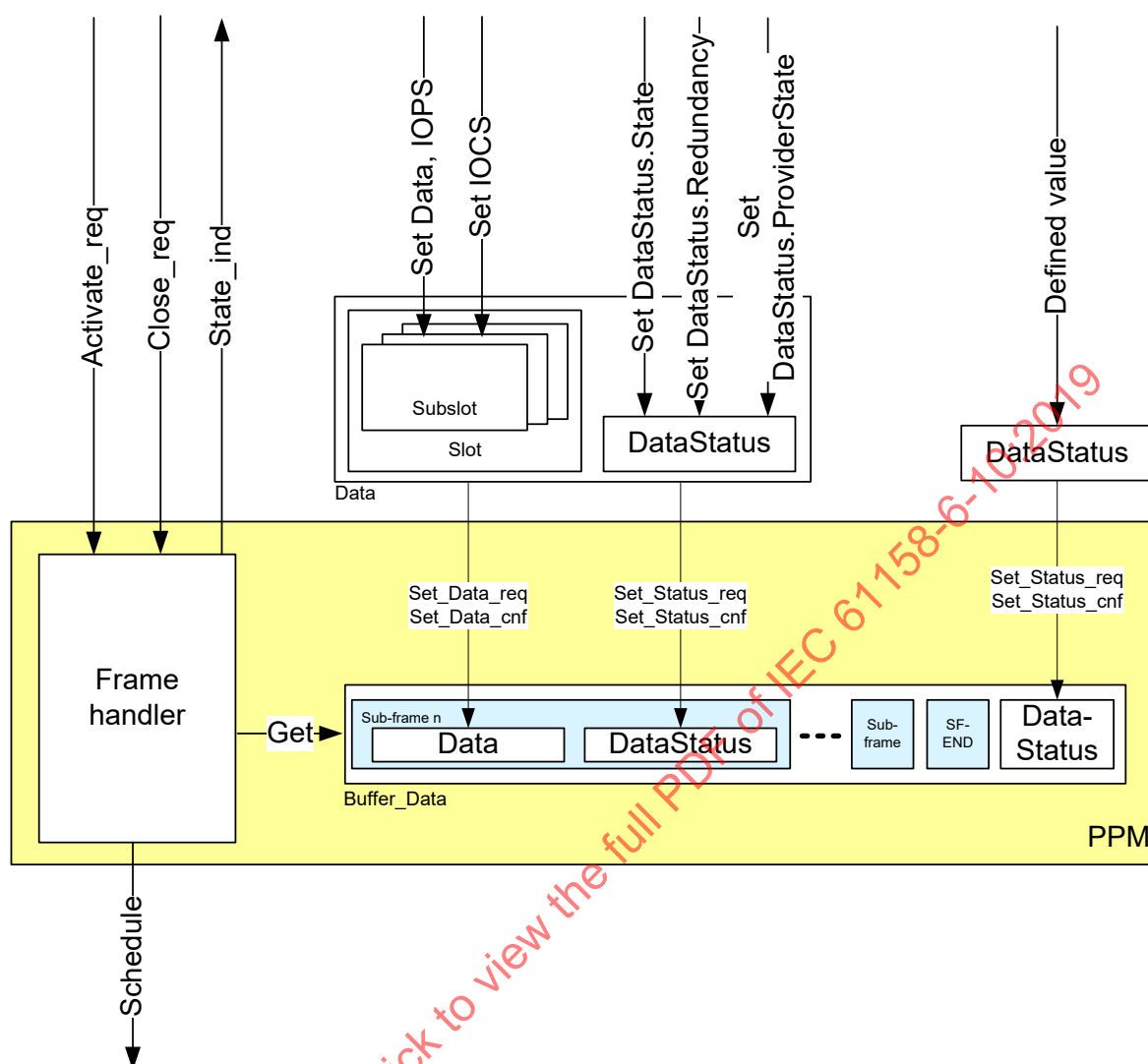


Figure 72 – Basic structure of a PPM with subframe structure

Table 207 – 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

4.7.4.1.2 Primitive definitions

4.7.4.1.2.1 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 208.

Table 208 – Remote primitives issued or received by PPM

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.7.4.1.2.2 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 209.

Table 209 – 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, Priority, 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	—

4.7.4.1.3 State transition diagram

The state transition diagram of the PPM is shown in Figure 73.

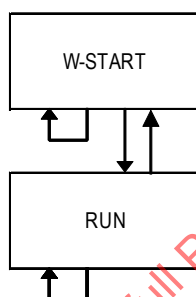


Figure 73 – State transition diagram of PPM

States of the PPM

W-START

Waiting for initialization and add the frame to the SCHEDULER

RUN

Running

4.7.4.1.4 State machine description

The W-START state indicates that an initialization is needed. The Activate service sets the machine in the RUN state waiting for Provider Data service requests and Time events. Receiving the confirmation will bring back the machine to the RUN state. An error or a Close service request is needed to re-enter the W-START state.

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.

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 and 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 IEEE 802.3 at the transmitting port.

NOTE The TagControlInformation.VID is set to zero by the PPM but can be set different elsewhere according to IEEE 802.1Q.

4.7.4.1.5 PPM state table

Table 210 contains the description of the PPM state table.

Table 210 – 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
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

#	Current State	Event /Condition =>Action	Next State
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

4.7.4.1.6 Functions, Macros, Timers and Variables

Table 211 contains the functions, macros, timers and variables used by the PPM and their arguments and their descriptions.

Table 211 – Functions, Macros, Timers and Variables used by the PPM

Name	Type	Function/Meaning
FirstTransmit	Variable	This local variable contains the information whether the frame was at least one time conveyed.
Buffer_Data	Variable	This local variable contains the data (encoded as DataItem*) which shall be used as storage of the data dedicated for conveyance of the next Data-RTC-PDU or UDP-RTC-PDU.
Buffer_Status	Variable	This local variable contains the status (encoded as APDU_Status.DataStatus) which shall be used as storage of the status dedicated for conveyance of the next Data-RTC-PDU or UDP-RTC-PDU.
NOTE DataItem* includes all DataItems for the dedicated PDU.		

4.7.4.1.7 Truth tables

Table 212 contains the valid combinations, their meaning and their description of TxOption.

Table 212 – Truth table used by the PPM for TxOption

RT_CLASS_X	ARProperties. StartupMode	IRT	MRPD	DFP	Description
		FrameSendOffset, Port	FrameID, FrameSendOffset, 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 Q)
RT_CLASS_3	Advanced	Values given	—	Values given	PPM shall create one frame (see Annex A)
RT_CLASS_3	Advanced	Values given	Values given	Values given	PPM shall create two frames (see Annex A and Annex Q)

4.7.4.2 Consumer protocol machine

4.7.4.2.1 General

Figure 74 shows the basic structure of the Consumer protocol machine (CPM).

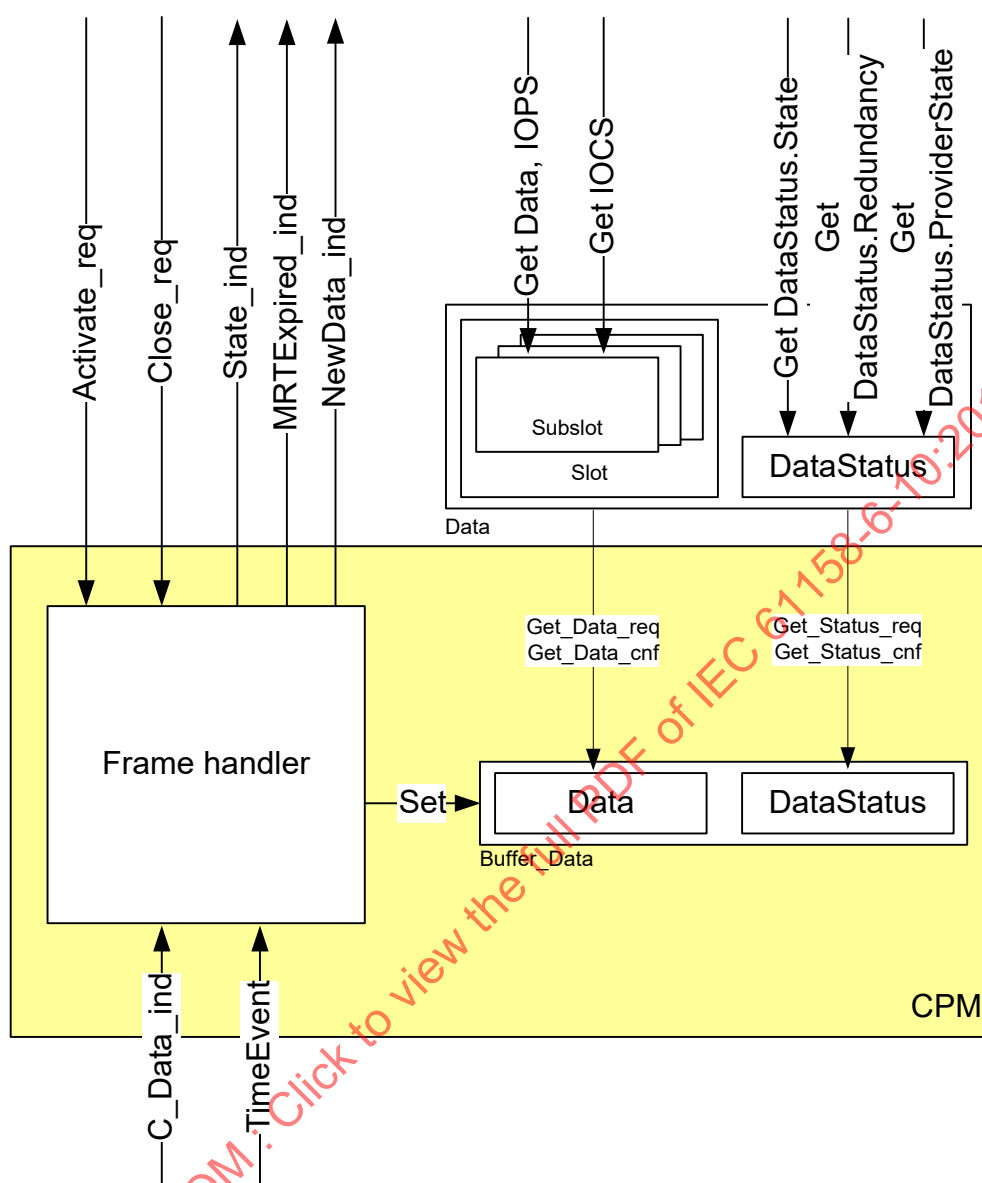


Figure 74 – Basic structure of a CPM

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 CPM are described in the service definition and shown in Table 213.

Table 213 – 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	—

4.7.4.2.2.2 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 214.

Table 214 – 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	CPM	CMSU CTLSU	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_MRTEExpired_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}	—

4.7.4.2.3 State transition diagram

The state transition diagram of the CPM is shown in Figure 75.

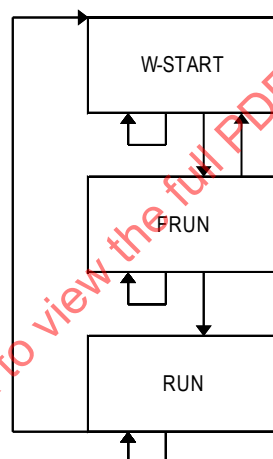


Figure 75 – State transition diagram of CPM

States of the CPM

W-START

The W-START state indicates that the initialization is needed. The Activate service sets the machine in the RUN state or in 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).

4.7.4.2.4 State machine description

This protocol state machine is the communication end point for the RT_CLASS_x cyclic conveyed data. It checks the received frames against the protocol and semantics and monitors the in time reception of the expected data.

Each time receiving valid C_Data.ind 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. For RT_CLASS_3 up to two frames with the same FrameID and for MRPD two frames with two FrameIDs (FrameID and FrameID+1) are handled.

4.7.4.2.5 CPM state table

Table 215 contains the complete description of the CPM state table.

Table 215 – CPM state table

#	Current State	Event /Condition =>Action	Next State
1	W-START	CPM_Activate_req () => Store SA, FrameID, RxOption, DataHoldFactor, Default_Value, Default_Status 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 ControllInterval:= SendClockFactor x ReductionRatio x 31,25 µs StartTimer (ControllInterval) 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
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 () =>	FRUN

#	Current State	Event /Condition =>Action	Next State
		Data:= Buffer_Data New_Flag:= New_Data New_Data:= FALSE CPM_Get_Data_cnf (+)	
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 (ControllInterval) => ignore	FRUN
14	FRUN	TimerExpired (MRWDt) => ignore	FRUN
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

#	Current State	Event /Condition =>Action	Next State
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 (ControllInterval) /Dht > (DataHoldFactor-1) => Dht:= 0 StopTimer () CPM_State_ind (Stop) //NOTE Dht expired	W- START
23	RUN	TimerExpired (ControllInterval) /Dht < (DataHoldFactor) => Dht:= Dht + 1.	RUN
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_MRTEexpired_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_MRTEexpired_ind (Fault) StartTimer (MRWDt)	RUN

4.7.4.2.6 Functions, Macros, Timers and Variables

Table 216 contains the functions, macros, timers and variables used by the CPM and their arguments and their descriptions.

Table 216 – Functions, Macros, Timers and Variables used by the CPM

Name	Type	Function/Meaning
Filter	Function	This local function filters from the received data the to be stored portion for the Buffer_Data.
FrameGuard	Function	This local function checks the received frame against the expected structure and semantics according 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
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
ControllInterval	Timer	This local timer contains the monitoring interval for the DataHoldFactor. ControllInterval:= SendClockFactor x ReductionRatio x 31,25 µs This timer shall use a autoreload function to start itself after it expires. It shall be aligned to the PTCP or local 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 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.

Name	Type	Function/Meaning
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 signalling 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 may 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 checking. 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}

Table 217 contains the valid combinations, their meaning and their description for RxOption.

Table 217 – Truth table used by the CPM for RxOption

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	FALSE	—	—	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	FALSE	Values given	—	Not supported
RT_CLASS_3	FALSE	—	Values given	Not supported
RT_CLASS_3	FALSE	Values given	Values given	Not supported
RT_CLASS_3	TRUE	—	—	CPM shall create one consumer for one frame (see Annex A)
RT_CLASS_3	TRUE	Values given	—	CPM shall create one consumer for two frames (see Annex A and Annex Q)
RT_CLASS_3	TRUE	—	Values given	CPM shall create one consumer for one frame (see Annex A)
RT_CLASS_3	TRUE	Values given	Values given	CPM shall create one consumer for two frames (see Annex A and Annex Q)

4.7.4.2.7 Truth tables

4.7.4.2.7.1 CPM checking rules for one frame

Table 218, Table 219, Table 220 and Table 221 contain the truth table used by the CPM for frames.

Table 218 – 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 219 – 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 220 – 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 190.
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, by this CPM, successful received and stored frame or APDU_Status.
^b See 4.7.2.1.2 and Table 190

Table 221 – 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 Combination used by the CTLSRL in combination with "UpdateInputsOnBackup == TRUE".

4.7.4.2.7.2 CPM checking rules for one subframe

Table 222, Table 223, Table 224 and Table 225 contain the truth table used by the CPM for subframes.

Table 222 – 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 223 – 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 224 – 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 225 – 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 226 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 226 shall represent the content of the DLSDU in Table 10.

Table 226 – RTA APDU syntax

APDU name	APDU structure
RTA-PDU	DATA-RTA-PDU ^ ACK-RTA-PDU ^ NACK-RTA-PDU ^ ERR-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 227 defines structures for substitutions of elements of the APDU structures shown in Table 226.

Table 227 – RTA substitutions

Substitution name	Structure
Reference	DestinationServiceAccessPoint, SourceServiceAccessPoint
Destination-ServiceAccess-Point	AlarmEndpoint
Source-ServiceAccess-Point	AlarmEndpoint
FlagsSequence	AddFlags, SendSeqNum, AckSeqNum
VendorDevice-ErrorInfo	VendorID, DeviceID, Data*
DATA-RTA-PDU	Reference, PDUType (=1), FlagsSequence, VarPartLen (1 – 1 432) ^a RTA-SDU
ACK-RTA-PDU	Reference, PDUType (=3), FlagsSequence, VarPartLen (=0)
NACK-RTA-PDU	Reference, PDUType (=2), FlagsSequence, VarPartLen (=0)
ERR-RTA-PDU	Reference, PDUType (=4), FlagsSequence, VarPartLen (=4), PNIOStatus, [VendorDeviceErrorInfo] ^b
^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 802.3	

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. It identifies, in combination with the Host-Address, the service access points of a bidirectional connection as shown in Figure 76. The values for source and destination shall be exchanged during a context management operation.

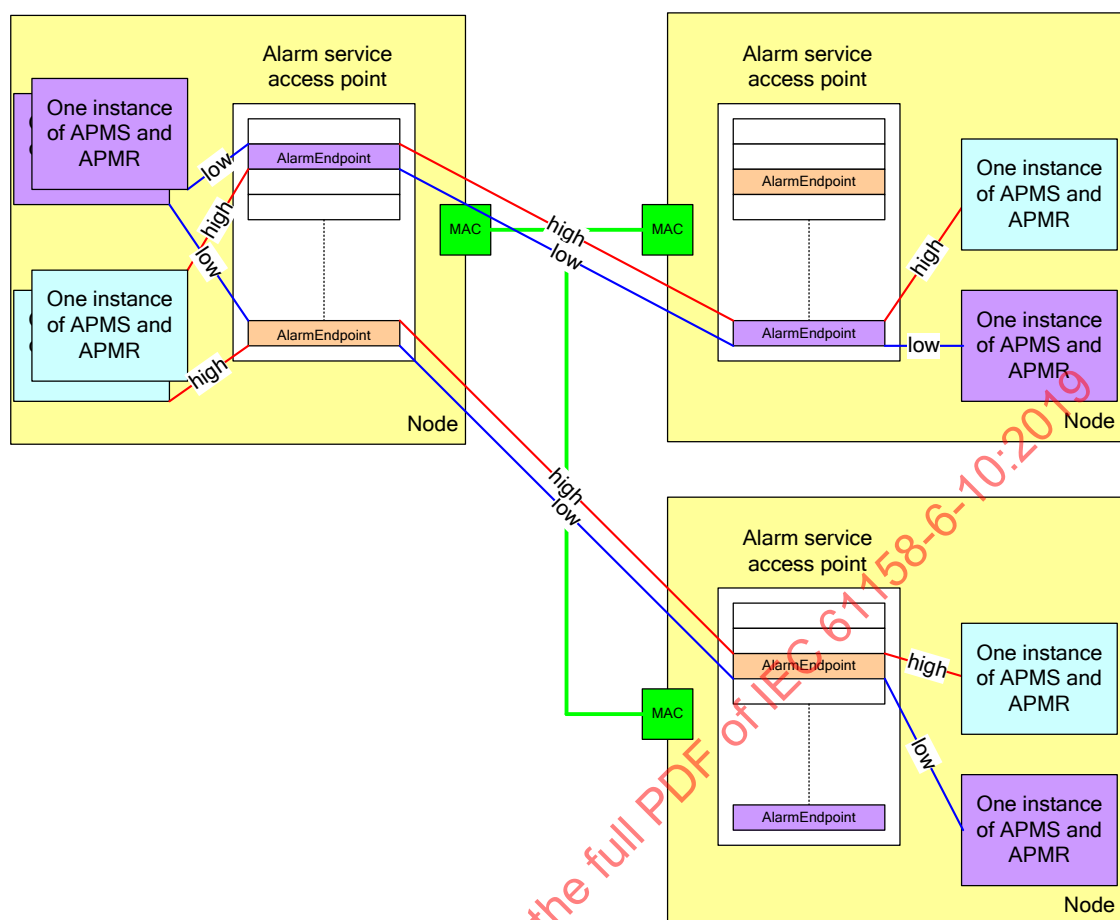


Figure 76 – Addressing scheme of RTA

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.

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 228.

Table 228 – PDUType.Type

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	—

Bit 4 – 7: PDUType.Version

This field shall be coded with the values according to Table 229.

Table 229 – PDUType.Version

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Version 1 of the protocol
0x02 – 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 – 3: AddFlags.WindowSize

This field shall be coded with the values according to Table 230.

Table 230 – AddFlags.WindowSize

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Window size one
0x02 – 0x07	Reserved

Bit 4: AddFlags.TACK

This field shall be coded with the values according to Table 231 by the sender within a RTA-PDU to control the acknowledge behavior of the receiver.

Table 231 – AddFlags.TACK

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 for DATA-RTA-PDU.

Bit 5 – 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 232.

Table 232 – SendSeqNum

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
0xFFFFE	This value shall be used to synchronize sender and receiver after establishment of the application relationship. Initial value which means “initial sequence, no acknowledge 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.

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 233 shows, as an example, the values during the initial synchronization between sender and receiver.

Table 233 – SendSeqNum and AckSeqNum start sequence

Service	Direction	Type	SendSeqNum	AckSeqNum
First AlarmNotification	IOD → IOC	DATA	0xFFFF	0xFFFFE
	IOD ← IOC	ACK	0xFFFFE	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

4.8.2.1.5 Coding of the field AckSeqNum

This field shall be coded as data type Unsigned16 with values according to Table 234.

Table 234 – AckSeqNum

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 acknowledge generated before”
0xFFFF	This value acknowledges the reception of the first DATA-RTA-PDU.

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 235. The VarPartLen shall not contain the DLPDU_Padding.

Table 235 – VarPartLen

Value (hexadecimal)	Meaning
0x0000	No RTA-SDU exists
0x0001 – 0x0598	A RTA-SDU with VarPartLen octets exists
0x0599 – 0xFFFF	Reserved

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 77. An ALPMI / ALPMR channel is addressed by Source-Node, Destination-Node, Source-SAP, Destination-SAP and FrameID high or low.

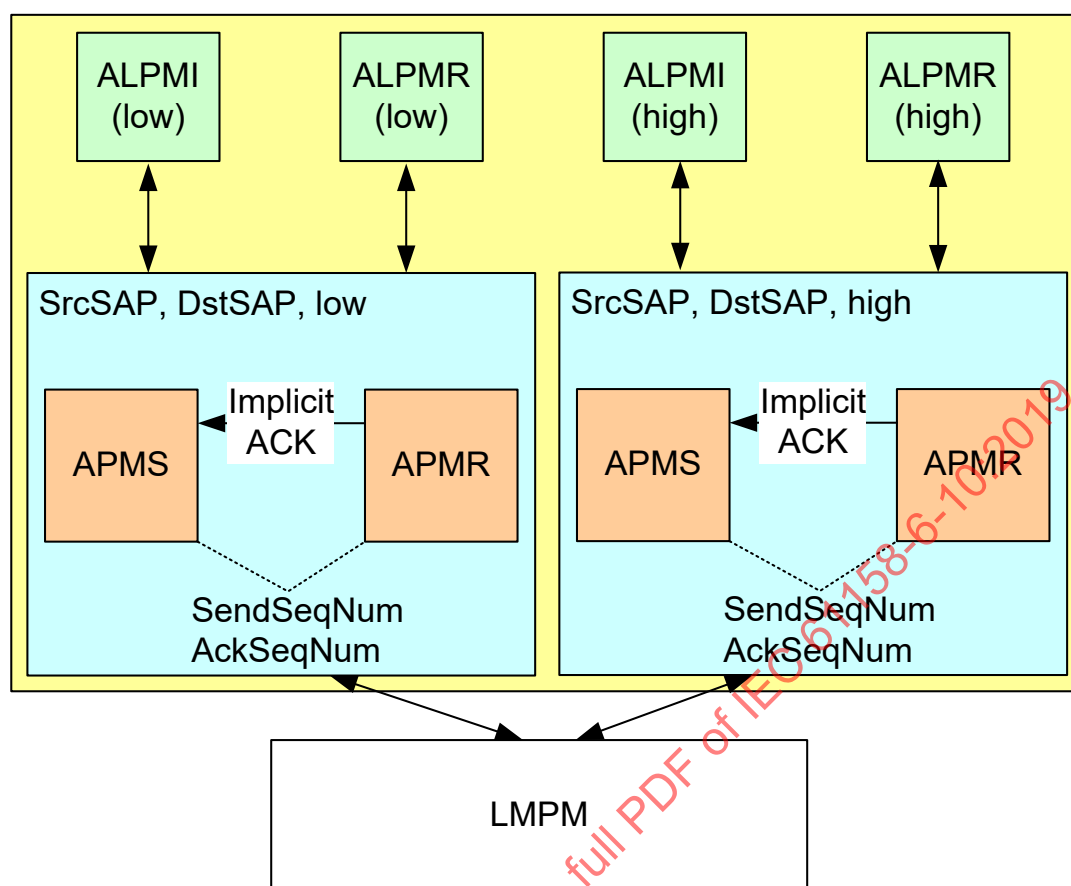


Figure 77 – Structure of the APM

4.8.4.2 Acyclic Protocol Machine Sender

4.8.4.2.1 General

In the following the structure of the Acyclic Protocol Machine Sender (APMS) is shown (see Figure 78).

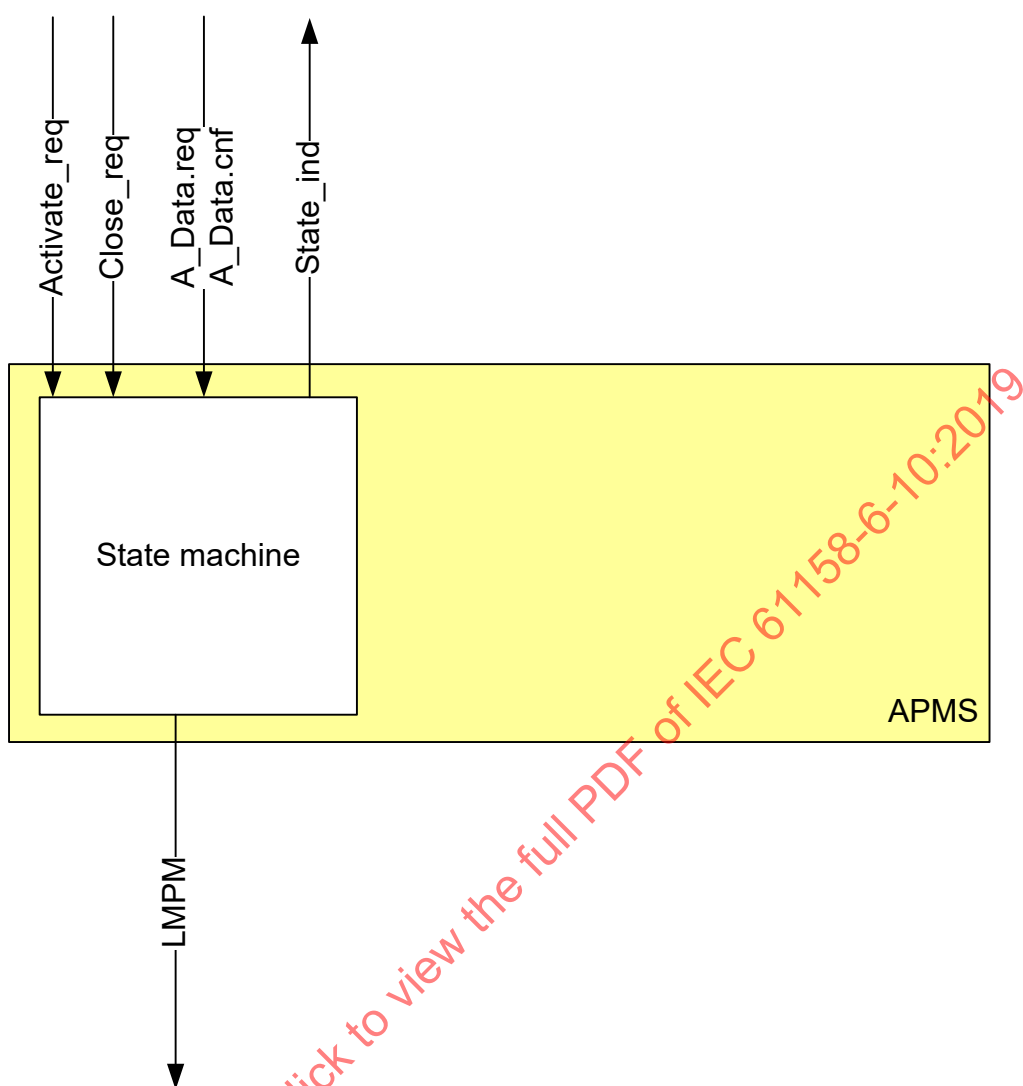


Figure 78 – Structure of the APMS

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 236.

Table 236 – 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	

Primitive	Source	Destination	Associated parameters	Functions
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 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 237.

Table 237 – 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 79.

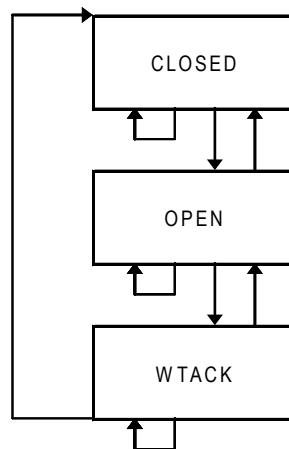


Figure 79 – State transition diagram of APMS

States of the APMS are:

CLOSED	The CLOSED state indicates that an initialization is needed. Activate service sets the machine in the OPEN state.
OPEN	Waiting for A-Data service request
WTACK	After issuing the transmission of data the WTACK state is used to wait for a transport acknowledge. Receiving the acknowledge will set back 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_VERSION=1.

The APMS state machine specifies the abort sequence and shows the corresponding ERR-RTA-PDU. These PDU shall only be sent from the APMS for low priority alarms from both sides of the connection.

A NAK doesn't count as a TACK and thus a retransmission takes place. If the retransmission is acknowledged again by NAKs, a timeout for TAK happens and an error shall be issued.

4.8.4.2.5 APMS state table

Table 238 contains the description of the APMS state machine.

Table 238 – 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
7	OPEN	APMS_Close.req () /VLANPrio == Low => 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 => //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

#	Current State	Event /Condition =>Action	Next State
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
15	WTACK	APMS_Close.req () /VLANPrio == Low => 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 => 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

#	Current State	Event /Condition =>Action	Next State
21	WTACK	A_Data_ind /PDU.PDUType == NAK => ignore //NOTE A NAK doesn't 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
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 239 contains the functions, macros, timers and variables used by the APMS, their arguments and their descriptions.

Table 239 – 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 is expired.
A_Data_cnf	Macro	if (Transport == RTC) 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 == RTC) 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 == RTC) 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)

Name	Type	Function/Meaning
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.

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 80).

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019

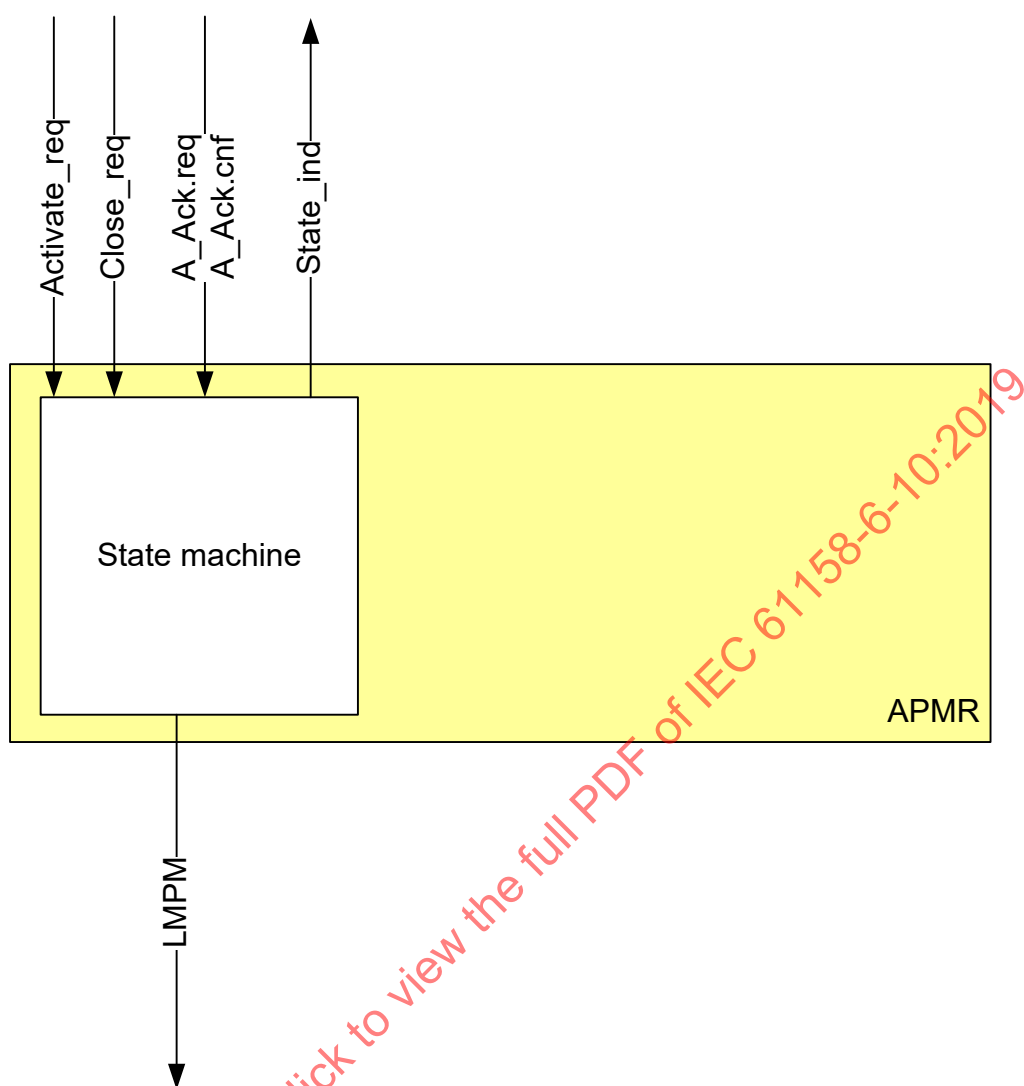


Figure 80 – Structure of the APMR

4.8.4.3.2 Primitive definitions

4.8.4.3.2.1 Primitives exchanged between remote machines

The service primitives including their associated parameters issued or received by APMR are described in the service definition and shown in Table 241.

Table 240 – 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 == Alarm-Ack-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

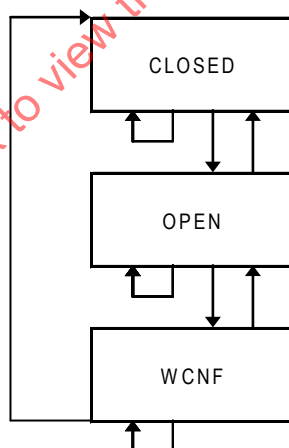
The local service primitives including their associated parameters issued or received by APMR are described in the service definition and shown in Table 241.

Table 241 – 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	—
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 81.

**Figure 81 – 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 in 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 independent. 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 242 contains the description of the APMR state machine.

Table 242 – 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:=0xFFFF 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

#	Current State	Event /Condition =>Action	Next State
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 with 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
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 doesn't 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

#	Current State	Event /Condition =>Action	Next State
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 243 contains the functions, macros, timers and variables used by the APMR and their arguments and their descriptions.

Table 243 – Functions, Macros, Timers and Variables used by the APMR

Name	Type	Meaning
A_Data_ind	Macro	if (Transport == RTC) 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 == RTC) 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 == RTC) 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.

4.8.5 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 on the lower end by the maximum size of an Ethernet frame. The fragmentation reduces, with a peer to peer approach, the conveyed size of an Ethernet frame and allows to use a LengthOfPeriod down to 31,25 µs.

It uses the IEEE 802.1Q defined TagControlInformation.PCP to encode and decode up to 8 (at least one shall be supported) parallel fragmentation streams. Because of the bundling to the IEEE 802.1Q the granularity of the supported streams shall be one, two, four, or eight and the assignment to the TagControlInformation.PCP is shown in Table 244.

Table 244 – TagControlInformation.PCP vs. streams

Priority	One stream	Two streams	Four streams	Eight streams
7	Default All priorities are assigned to one stream	The upper four priorities are assigned to one stream	Two priorities are assigned to one stream	One priority is one stream
6				One priority is one stream
5			Two priorities are assigned to one stream	One priority is one stream
4				One priority is one stream
3		The lower four priorities are assigned to one stream	Two priorities are assigned to one stream	One priority is one stream
2				One priority is one stream
1			Two priorities are assigned to one stream	One priority is one stream
0				One priority is one stream

Even for the frames, generated by the fragmentation, the lower frame size limit of IEEE 802.3 (64 octets without VLAN and 68 octets with VLAN) applies. Thus limits the use of the principle of fragmentation according to Figure O.2, Formula (42), Formula (43), Formula (44) and Formula (45) as shown in Table 245.

The following Formula (42) 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.

$$\text{LowestFractionizeableFrameSize} = \text{FrameDataFromFirstFragment} + \text{FrameDataFromLastFragment} + \text{ProtocolOverhead} \quad (42)$$

With VLAN

$$\text{LowestFractionizeableFrameSize} = 40 \text{ octets} + 38 \text{ octets} + 20 \text{ octets}$$

$$\text{LowestFractionizeableFrameSize} = 98 \text{ octets}$$

Without VLAN

$$\text{LowestFractionizeableFrameSize} = 48 \text{ octets} + 42 \text{ octets} + 16 \text{ octets}$$

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 (43) shows the calculation for the minimum leftover 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.

$$\text{LowestFractionizeableFrameSize} = \text{FrameDataFromMiddleFragment} + \text{FrameDataFromLastFragment} + \text{ProtocolOverhead} \quad (43)$$

With VLAN

$$\text{LowestFractionizeableFrameSize} = 40 \text{ octets} + 38 \text{ octets} + 26 \text{ octets}$$

$$\text{LowestFractionizeableFrameSize} = 104 \text{ octets}$$

Without VLAN

$$\text{LowestFractionizeableFrameSize} = 48 \text{ octets} + 42 \text{ octets} + 22 \text{ octets}$$

$$\text{LowestFractionizeableFrameSize} = 112 \text{ octets}$$

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 (44) shows a frame used for the first, middle or a last fragment with a VLAN tag.

$$\text{Lowest frame size(with VLAN)} = \text{DestinationMAC} + \text{SourceMAC} + \text{VLAN} + \text{FragmentationHeader} + \text{Payload} + \text{FCS} \quad (44)$$

$$\text{Payload(with VLAN)} = \text{Lowest frame size} - (\text{DestinationMAC} + \text{SourceMAC} + \text{VLAN} + \text{FragmentationHeader} + \text{FCS})$$

$$\text{Payload(with VLAN)} = 64 - (6 + 6 + 4 + 6 + 4)$$

$$\text{Payload(with VLAN)} = 38$$

according the rules for FragStatus.MoreFollows:=More fragments follows

$$\text{Payload(with VLAN)} = 40$$

according the rules for FragStatus.MoreFollows:=Last fragment

$$\text{Payload(with VLAN)} = 38$$

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 (45) shows a frame used for the first, middle or a last fragment without a VLAN tag.

$$\text{Lowest frame size(without VLAN)} = \text{DestinationMAC} + \text{SourceMAC} + \text{FragmentationHeader} + \text{Payload} + \text{FCS} \quad (45)$$

$$\text{Payload(without VLAN)} = \text{Lowest frame size} - (\text{DestinationMAC} + \text{SourceMAC} + \text{FragmentationHeader} + \text{FCS})$$

$$\text{Payload(without VLAN)} = 64 - (6 + 6 + 6 + 4)$$

$$\text{Payload(without VLAN)} = 42$$

according the rules for FragStatus.MoreFollows:=More fragments follows

$$\text{Payload(without VLAN)} = 48$$

according the rules for FragStatus.MoreFollows:=Last fragment

$$\text{Payload(without VLAN)} = 42$$

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 245 – 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 kind 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 246 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 246 shall represent the content of the DLSDU in Table 10.

Table 246 – FRAG APDU syntax

APDU name	APDU structure
FRAG-PDU	FRAG-DATA-PDU

Table 247 defines structures for substitutions of elements of the APDU structures shown in Table 246.

Table 247 – 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 “leftover” octets may be added.

FragDataLength shall be calculated according to Formula (46).

$$\text{FragDataLength} = (\text{Length of FragData} \text{ } \textit{DIV} \text{ } 8) \quad (46)$$

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 leftover, then the size of the last fragment containing the FragData shall be the rest, containing the leftover 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 248.

Table 248 – 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 leftover 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 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 249.

Table 249 – FragStatus.FragmentNumber

Value (hexadecimal)	Meaning
0x00	First fragment of a frame
0x01	Second fragment of a frame
...	...
0x3F	64 th fragment of a frame

Bit 6: FragStatus.Reserved

This field shall be coded according to Table 250.

Table 250 – FragStatus.Reserved

Value (hexadecimal)	Meaning
0x00	Reserved

Bit 7: FragStatus.MoreFollows

This field shall be coded according to Table 251.

Table 251 – FragStatus.MoreFollows

Value (hexadecimal)	Meaning
0x00	Last fragment
0x01	More fragments follows

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 service primitives including their associated parameters issued or received by FRAG_D are described in the service definition and shown in Table 252.

Table 252 – 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 253.

Table 253 – 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 82.

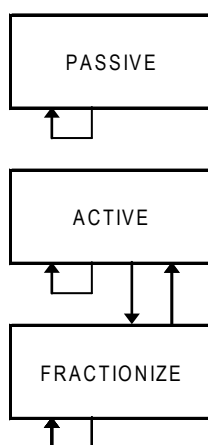


Figure 82 – State transition diagram of FRAG_D

States of the FRAG_D are:

PASSIVE	The FRAG unit is not used. Frames passes 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 for 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 254 contains the FRAG_D state table.

Table 254 – 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 255 contains the functions, macros, timers and variables used by the FRAG_D and their arguments and their descriptions.

Table 255 – 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 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 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 802.3 frame are still valid, even for fragments.
FrameFitsIntoResidualPeriod	Function	This local function checks whether the residual GREEN period is longer than 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 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 an 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 ends the FRAG stream.
Period	Macro	This local macro contains the information what 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 service primitives including their associated parameters issued or received by FRAG_S are described in the service definition and shown in Table 252.

Table 256 – 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 254.

Table 257 – 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 83.

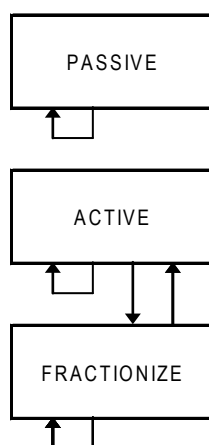


Figure 83 – State transition diagram of FRAG_S

States of the FRAG_S are:

PASSIVE	The FRAG unit is not used. Frames passes 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 for 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.

4.9.6.1.2.4 FRAG_S state table (static fragmentation)

Table 258 contains the FRAG_S state table.

Table 258 – 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
7	FRACTIONIZE	StartFragStream () => CalculateFragSize () CreateAllFrag () 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

#	Current State	Event /Condition =>Action	Next State
10	FRACTIONIZE	QueuesNotEmpty () /CheckForFrameWithHigherPriority () == EXISTS && FrameSizeAllowsFragmentation == TRUE => ignore	FRACTIONIZE

4.9.6.1.2.5 Functions, Macros, Timers and Variables

Table 259 contains the functions, macros, timers and variables used by the FRAG_S and their arguments and their descriptions.

Table 259 – 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 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 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 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 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 if the last fragment is conveyed.
Period	Macro	This local macro contains the information what 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 service primitives including their associated parameters issued or received by DEFRAG are described in the service definition and shown in Table 260.

Table 260 – 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

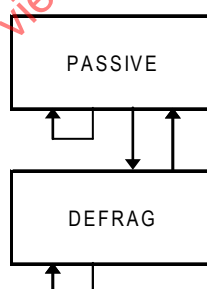
The local service primitives including their associated parameters issued or received by DEFRAG are described in the service definition and shown in Table 261.

Table 261 – 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 84.

**Figure 84 – State transition diagram of DEFRAG**

States of the DEFRAG are:

- PASSIVE** Waiting for the first fragment of a stream. For each supported stream 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 802.3, the defragmentation is aborted and the state PASSIVE entered.

4.9.6.2.3 State machine description

A DEFRAG state machine shall exist for every 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 is active.

4.9.6.2.4 DEFRAG state table

Table 262 contains the DEFRAG state table which exists for each port.

Table 262 – 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
8	DEFRAG	DMUX_DEFRAG.ind () /DefragGuard () == Invalid => Delete stored fragments	PASSIVE

4.9.6.2.5 Functions, Macros, Timers and Variables

Table 263 contains the functions used by the DEFRAG, their arguments and their descriptions.

Table 263 – 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 store 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 out of the stored fragments
FrameLimitReached	Macro	This local macro checks whether the IEEE 802.3 frame limits are exceeded.

4.9.6.2.6 DefragGuard

The DefragGuard checks the received frame against Table 264, Table 265 and Table 266. The used table depends on the local supported functionality.

The defragmentation shall accept a frame independent from the value of FrameID.FragSequence in combination with FragStatus.FragmentNumber == 0 as start of a new defragmentation sequence.

Table 264 – 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 265 – 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 store one from the first fragment

^b The stored one from the last received fragment of this FragSequence (FragmentNumber == 1)

Table 266 – 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 267 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 267 shall represent the content of the DLSDU in Table 10.

Table 267 – RPC APDU syntax

APDU name	APDU structure
CL-RPC-PDU	IPHeader, UDPHeader, RPCHeader, NDRData
NDRData	NDRDataRequest ^ NDRDataResponse ^ NDREMapLookupReq ^ NDREMapLookupRes ^ NDREMapLookupFreeReq ^ NDREMapLookupFreeRes ^ NDRAck ^ NDRQuAck ^ NDRQuit ^ NDRFack ^ NDRFault ^ NDRNoCall ^ NDRWorking ^ NDRPing ^ NDRReject

Table 268 defines structures for substitutions of elements of the APDU structures shown in Table 267.

Table 268 – 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	RPCNCARrejectStatus
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*])], RPCEMMapStatus
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
NDREPMAPLookupFreeReq	RPCEntryHandleAttribute, RPCEntryHandleUUID
NDREPMAPLookupFreeRes	RPCEntryHandleAttribute, RPCEntryHandleUUID, RPCEMMapStatus
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 269.

Table 269 – 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 270.

Table 270 – 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 271.

Table 271 – 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 272.

Table 272 – RPCFlags2

Bit	Meaning if set to 1
Bit 0	Implementation specific, shall be set to 0
Bit 1	Cancel was pending at call end
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 273.

Table 273 – 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 NDREPMMap substitutions within Type 10. It only uses Unsigned8, Unsigned16, or Unsigned32 there. The user data of the Type 10 service definitions are not influenced 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 274.

Table 274 – 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 275.

Table 275 – RPCObjectUUID.Data4

Octet	Meaning
1	Values see Table 276
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 276 and Table 277. These values shall be used in conjunction with the values from Table 278.

Table 276 – 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 instance, where
xxxx	Represents the instance or node number See Table 277
yyyy	Identify the Device ID as a vendor specific number for the device class
zzzz	Represents the Vendor ID as a central administrative number assigned by the responsible user organisation

Table 277 – Instance or node number

Value	Description
xxxx	Identifies an object instance within a physical device in case there are more than one instance, where
Bit 0 – Bit 11	Lower part of the instance or node number
Bit 12 – Bit 15	Identify the interface 0x0: Interface 0 (Default) ... 0xF: Interface 15 See "I" within Formula (49)

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 268.

Defined values for this document are shown in Table 278.

Table 278 – 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

Defined values for the RPC end point mapper are shown in Table 279.

Table 279 – RPCInterfaceUUID for the RPC end point 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 endianness is according to the value in the field “RPCDRep.IntegerEncoding” (little endian or big endian).

The IOC and the IOD generate an RPCActivityUUID for each AR and use them as long as the AR exist.

Any RPC client continuously using Implicit Read with an IOD or IOC shall generate one RPCActivityUUID for this communication relation and use them as long as this communication relation exists.

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 endpointmapper 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 280.

Table 280 – 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 281.

Table 281 – RPCInterfaceVersion.Minor

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Valid version information

The endianness is according to the value in 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 a 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

- IO device,
- IO controller and
- IO supervisor.

The values for PNIO services are defined in Table 282.

Table 282 – RPCOperationNmb (IO device, controller and supervisor)

Value (decimal)	Service	Usage
0	Connect	—
1	Release	—
2	Read	Only valid with ARUUIID<>0
3	Write	Only valid with ARUUIID<>0
4	Control	—
5	Read Implicit	Only valid with ARUUIID=0
6 – 65 535	Reserved	—

The values for endpoint mapper services according to Table 283 shall be used.

Table 283 – 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 284.

Table 284 – 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:

- 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 268.

The value for this document is defined in Table 285.

Table 285 – RPCDataRepresentationUUID – defined values

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 NDREPMAPDU

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 manufacturer specific 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 manufacturer specific 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 character "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 character:

- "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 286.

Table 286 – 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 268.

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 268.

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 0.

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 0.

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 0.

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 0.

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 287.

Table 287 – 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 on the server side and an error should be reported locally on 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 the following formula.

$$\begin{aligned} \text{PNIOStatus} = & \text{ErrorCode} \times 16\,777\,216 + \\ & \text{ErrorDecode} \times 65\,536 + \\ & \text{ErrorCode1} \times 256 + \\ & \text{ErrorCode2} \end{aligned} \quad (47)$$

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 288.

Table 288 – 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

Value (hexadecimal)	Definition
0x1C00000A	NCA_s_bad_actid
0x1C00000B	NCA_s_who_are_you_failed
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 289.

Table 289 – 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

Value (hexadecimal)	Definition
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 service primitives including their associated parameters issued or received by RPC are described in the service definition and shown in Table 290.

Table 290 – 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 291.

Table 291 – 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 don't care.

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 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 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 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 292 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 292 shall represent the content of the DLSDU in Table 10.

Table 292 – LLDP APDU syntax

APDU name	APDU structure
LLDP-PDU	LLDPChassis, LLDPPort, LLDP TTL, LLDP-PNIO-PDU, LLDPEnd
LLDP-PNIO-PDU	{ [LLDP_PNIO_DELAY] ^a , LLDP_PNIO_PORTSTATUS, [LLDP_PNIO_ALIAS], [LLDP_PNIO_MRPPORTSTATUS] ^b , [LLDP_PNIO_MRPICPORTSTATUS] ^g , LLDP_PNIO_CHASSIS_MAC, LLDP8023MACPHY ^d , LLDPManagement, [LLDP_PNIO_PTCPSTATUS] ^c , [LLDP_PNIO_MAUTypeExtension] ^f , [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 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.	

Table 293 defines structures for substitutions of elements of the APDU structures shown in Table 292.

Table 293 – 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
LLDP TTL	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_MRPICPORTSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x08), MRPIC_DomainID, MRPIC_Role
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
LLDPEnd	LLDP_TLVHeader ^b (0)

Substitution name	Structure
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
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 802.1AB-2016. ^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 zero, if unknown. ^f Shall be PTCP_SubdomainUUID of PTCP_SyncID = 0. Otherwise the value shall be zero. ^g Shall be set zero, if unknown. ^h Shall be set according to IEEE 802.1AB-2016, Annex E. ⁱ Shall be set according to IEEE 802.1AB-2016, Annex F. ^j Shall be set according to IEEE 802.1AB-2016, 8.5.9. It is recommended to set the object identifier according to 4.16.3 or with a vendor specific MIB. If both are not implemented, than the "object identifier (OID) string length" shall be set to zero.	
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_ChassisID

This field shall be coded as data type OctetString according to Table 294 and Table 295. The content of this field shall uniquely identify one device, as defined by IEEE 802.1AB.

NOTE The field LLDP_ChassisID is not terminated by zero.

**Table 294 – LLDP_ChassisID in conjunction with
MultipleInterfaceMode.NameOfDevice == 0 and NameOfStation**

NameOfStation exist	Data type	Meaning
Yes	OctetString[]	This field shall be coded with 1 to 240 octets according to 4.3.1.4.15.
No	OctetString[]	This field shall be coded according LLDPChassisMacAddress in Table 293 or Table 295.

**Table 295 – 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 294 even if MultipleInterfaceMode.NameOfDevice == 1 is set.

4.11.3.2 Coding of the field LLDP_PortID

This field shall be coded as data type OctetString according to Table 296.

NOTE The field LLDP_PortID is not terminated by zero.

Table 296 – 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 4.3.1.4.16
1	OctetString[]	The field LLDP_PortID shall be coded according to Formula (48)

$$\text{LLDP\_PortID} = \text{NameOfPort} + "." + \text{NameOfStation} \quad (48)$$

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

4.11.3.3 Coding of the field LLDP_PNIO_SubType

This field shall be coded as data type Unsigned8 and shall be set according to Table 297.

Table 297 – 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 – 0xFF	Reserved

4.11.3.4 Coding of the field PTCP_PortRxDelayLocal

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 298.

Table 298 – PTCP_PortRxDelayLocal

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Local RX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.11.3.5 Coding of the field PTCP_PortRxDelayRemote

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 299.

Table 299 – PTCP_PortRxDelayRemote

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Remote RX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.11.3.6 Coding of the field PTCP_PortTxDelayLocal

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 300.

Table 300 – PTCP_PortTxDelayLocal

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Local TX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.11.3.7 Coding of the field PTCP_PortTxDelayRemote

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 301.

Table 301 – PTCP_PortTxDelayRemote

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Remote TX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.11.3.8 Coding of the field CableDelayLocal

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 302.

Table 302 – CableDelayLocal

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x000FFFFF	Measured cable delay
0x00100000 – 0xFFFFFFFF	Reserved

4.11.3.9 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 303 or Table 304.

Table 303 – RTClass2_PortStatus.State with ARProperties.StartupMode == Legacy

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used or configured if LLDP_OrangePeriodBegin.Valid is equal one.
0x01	Reserved	—
0x02	RTCLASS2_RUN	ORANGE Phase activated for transmission and reception of RTClass2 Frames
0x03	Reserved	—

Table 304 – RTClass2_PortStatus.State with ARProperties.StartupMode == Advanced

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.

4.11.3.10 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 305.

Table 305 – RTClass3_PortStatus.State

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used or configured if LLDP_RedOrangePeriodBegin.Valid is equal one. RED period is deactivated.
0x01	Reserved	—
0x02	RTCLASS3_UP	RED period activated for transmission of RTClass3 Frames. Expected next state: RTCLASS3_RUN
0x03	Reserved	—
0x04	RTCLASS3_RUN	RED period activated for transmission and reception of RTClass3 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 306.

Table 306 – 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 307.

Table 307 – 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 308 contains the truth table used for the shortening of the preamble.

Table 308 – Truth table for shortening of the preamble

Input		Output
RTC3PSM	AdjustPreambleLength. PreambleLength.Length	RTCClass3_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
! (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 309.

Table 309 – RTClass3_PortStatus.Optimized

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	ON	Reserved

4.11.3.11 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 310.

Table 310 – 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.

4.11.3.12 Coding of the field IRDataUUID

This field shall be coded as data type UUID according to Table 311.

Table 311 – 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-FFFFFFFFFFFFFFE	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

4.11.3.13 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 35. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_RedOrangePeriodBegin.Offset

This field shall be set according to Table 312.

Table 312 – 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 313.

Table 313 – 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.

4.11.3.14 Coding of the field LLDP_OrangePeriodBegin

NOTE Used only with ARProperties.StartupMode == Legacy

The coding of this field shall be according to 3.4.2.3.5 and to Figure 35. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_OrangePeriodBegin.Offset

This field shall be set according to Table 314.

Table 314 – LLDP_OrangePeriodBegin.Offset

Value (hexadecimal)	Meaning	Usage
0x00000000 – 0x003D08FF	Offset relative to the begin of the cycle in nanoseconds	Begin of the RT_CLASS_2 period of the transmit direction of the port. If a RedPeriod is defined, the value shall be derived from the maximum of OrangePeriodBegin, else the value shall be derived from ReservedIntervalBegin.
0x003D0900 – 0x7FFFFFFF	Reserved	—

Bit 31: LLDP_OrangePeriodBegin.Valid

This field shall be set according to Table 315 or Table 316.

Table 315 – LLDP_OrangePeriodBegin.Valid with ARProperties.StartupMode == Legacy

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	The value of LLDP_OrangePeriodBegin.Offset is not valid. It shall be set to zero.
0x01	Valid	The value of LLDP_OrangePeriodBegin.Offset is valid.

Table 316 – LLDP_OrangePeriodBegin.Valid with ARProperties.StartupMode == Advanced

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	LLDP_OrangePeriodBegin.Offset is not valid. It shall be set to zero.
Other	—	Reserved

4.11.3.15 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 35. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_GreenPeriodBegin.Offset

This field shall be set according to Table 317.

Table 317 – 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 318.

Table 318 – 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.

4.11.3.16 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 to Figure 35. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_LengthOfPeriod.Length

This field shall be set according to Table 319.

Table 319 – 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 320.

Table 320 – 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.

4.11.3.17 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.

4.11.3.18 Coding of the field LLDP_TimeToLive

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.19 Coding of the field LLDP_TLVHeader

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.20 Coding of the field LLDP_ChassisIDSubType

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.21 Coding of the field LLDP_PortIDSubType

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.22 Coding of the field LLDPOption

This field shall be coded according to IEEE 802.1AB-2016, 9.4.

4.11.3.23 Coding of the field LLDP_OUI

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.24 Coding of the field LLDP_8021_SubType

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.25 Coding section related to LLDP_Management

4.11.3.25.1 Coding of the field LLDP_ManagementData

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.26 Coding section related to LLDP_8023

4.11.3.26.1 Coding of the field LLDP_8023_SubType

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.26.2 Coding of the field LLDP_8023_AUTONEG

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.26.3 Coding of the field LLDP_8023_PMDCAP

This field shall be coded according to IEEE 802.1AB-2016.

4.11.3.26.4 Coding of the field LLDP_8023_OPMAU

This field shall be coded according to IEEE 802.1AB-2016.

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 Bridges and End Stations

4.12.1 General

The concepts according to following standards shall be applied:

- IEEE 802.3
- IEEE 802.1AB
- IEEE 802.1AS
- IEEE 802.1Q
- This document

The using of the “Cut through” principle for the forwarding of frames needs a late error handling to avoid frame fragments in a switched network. Thus, whenever an invalid (by FCS or SFCRC16) frame is detected which is already partial forwarded, it should be shortened according to the rules of IEEE 802.3 at the transmitting port.

According to this concept, every Bridge shall not forward frames where the SourceMACAddress is its own one and shall support wire speed forwarding of frames.

Figure 85 shows the integration of IEEE 802 into the model used in this document.

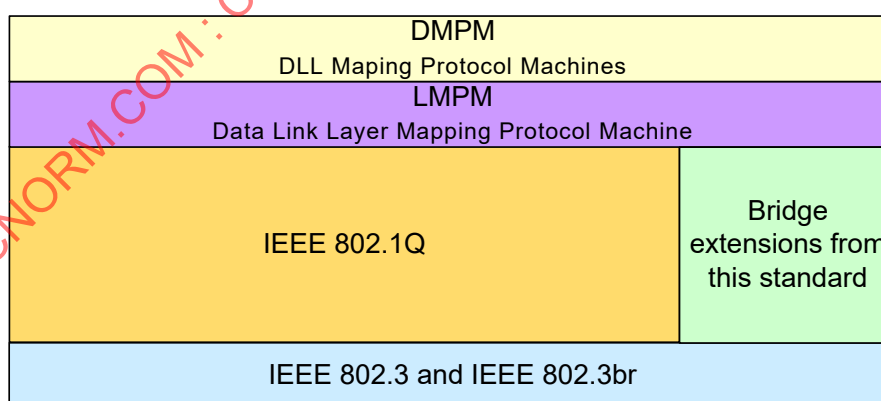


Figure 85 – DLL Mapping Protocol Machines (DMPM)

The IEEE 802.1Q added time awareness to its working principle and thus, extended its bridge model to the end station.

This document specifies the following three kinds of devices:

- one port bridges (End station),
- two port bridges,

– n port bridges.

4.12.2 Model

4.12.2.1 Cross reference

4.12.2.1.1 Bridge Port Transmit and Receive

4.12.2.1.1.1 General

See IEEE 802.1Q chapter 8.5 and chapter 6.9.

If a port is defined as Boundary Port for a domain, then this module executes tagging and detagging.

These frames are tagged on receive and detagged 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

4.12.2.1.1.2 Non time aware bridge

Used as specified by IEEE 802.1Q.

4.12.2.1.2 Priority Regeneration Override

4.12.2.1.2.1 General

See IEEE 802.1Q chapter 12.20.3.

If a port is defined as Boundary Port for a domain, then this module executes Priority Regeneration Override. Additional, the frame will be tagged with the VID of the domain, if untagged.

Peer to peer protocols like IEEE 802.1AS or IEEE 802.1AB are not influenced by this module.

4.12.2.1.2.2 Non time aware bridge

4.12.2.1.2.2.1 Without RT_CLASS_3

Used as specified by IEEE 802.1Q.

4.12.2.1.2.2.2 With RT_CLASS_3

The priority of a received frame will be overridden according to Table 321.

Table 321 – Priority remapping at an ingress boundary port

TCI.PCP Source	->	TCI.PCP Destination
7	unchanged	7
6		6
5		5
4		4
3	is overridden t	2
2	unchanged	2
1		1
0 or untagged		0 or untagged

4.12.2.1.3 Reception Port State

See IEEE 802.1Q chapter 8.13.9 and IEEE 802.1AC.

Two flags are used for port control:

- MAC_Operational (See IEEE 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 802.1AS or
- IEEE 802.1AB or
- VID,

defines its own tree related reception port state.

4.12.2.1.4 Transmission Port State

See IEEE 802.1Q chapter 8.13.9 and IEEE 802.1AC

Two flags are used for port control:

- MAC_Operational (See IEEE 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 802.1AS or
- IEEE 802.1AB or
- VID,

defines its own tree related reception port state.

4.12.2.1.5 Filtering Database

4.12.2.1.5.1 General

See IEEE 802.1Q chapter 8.8

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.

4.12.2.1.5.2 Non time aware bridge

Supported trees are

- Default
- IEC 62439-2
- IEEE 802.1AS Working Clock
- IEEE 802.1AS Global Time
- IEEE 802.1AB
- PTCP
- RT_CLASS_3

Table 322 shows whether the trees are with a FDB or assigned to LLC.

Table 322 – Trees and FDBs

Tree	Assignment	Learning	Default Forwarding Rule
Default VID	FDB "Default"	Enabled, IVL	Flooding
IEC 62439-2	FDB "Default" and/or LLC	Disabled	Dropping
IEEE 802.1AS Working Clock	LLC	Disabled	—
IEEE 802.1AS Global Time Redundant	LLC	Disabled	—
IEEE 802.1AB	LLC	Disabled	—
PTCP	LLC	Disabled	—
RT_CLASS_3	FDB "RED_RELAY"	Disabled	Dropping

4.12.2.1.6 Active topology enforcement

4.12.2.1.6.1 General

See IEEE 802.1Q chapter 8.6.1

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

4.12.2.1.7 Ingress

See IEEE 802.1Q chapter 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.

4.12.2.1.8 Frame filtering

See IEEE 802.1Q chapter 8.6.3

Apply the rules from the found tree and its assigned FDB to the frame to decide about the transmission port or ports.

4.12.2.1.9 Egress

See IEEE 802.1Q chapter 8.6.4

Check if this transmit port is part of an identified tree. Drop frame if VID is unknown.

4.12.2.1.10 Flow classification and metering

4.12.2.1.10.1 General

See IEEE 802.1Q chapter 8.6.5

Ingress rate limiting based on destination MAC address.

The Ingress Rate Limiter, defined flow classification and metering, allows an ingress flow control to protect the resources of a bridge.

4.12.2.1.10.2 Non time aware bridge

Feature not used.

4.12.2.1.11 Queuing frames

4.12.2.1.11.1 General

4.12.2.1.11.1.1 General

See IEEE 802.1Q chapter 8.6.6

Identify the traffic class and thus, the queue in which the frame will be stored. Table 323 shows the available queues.

Each frame shall be assigned one queue. This is done by TCI.PCP or in case of MAC service, as defined in IEEE 802.1AC, directly.

Table 323 – Available queue

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

4.12.2.1.11.1.2 Model

Each Queue Frame entity of a bridge implements a resource management following the principle shown in Figure 86.

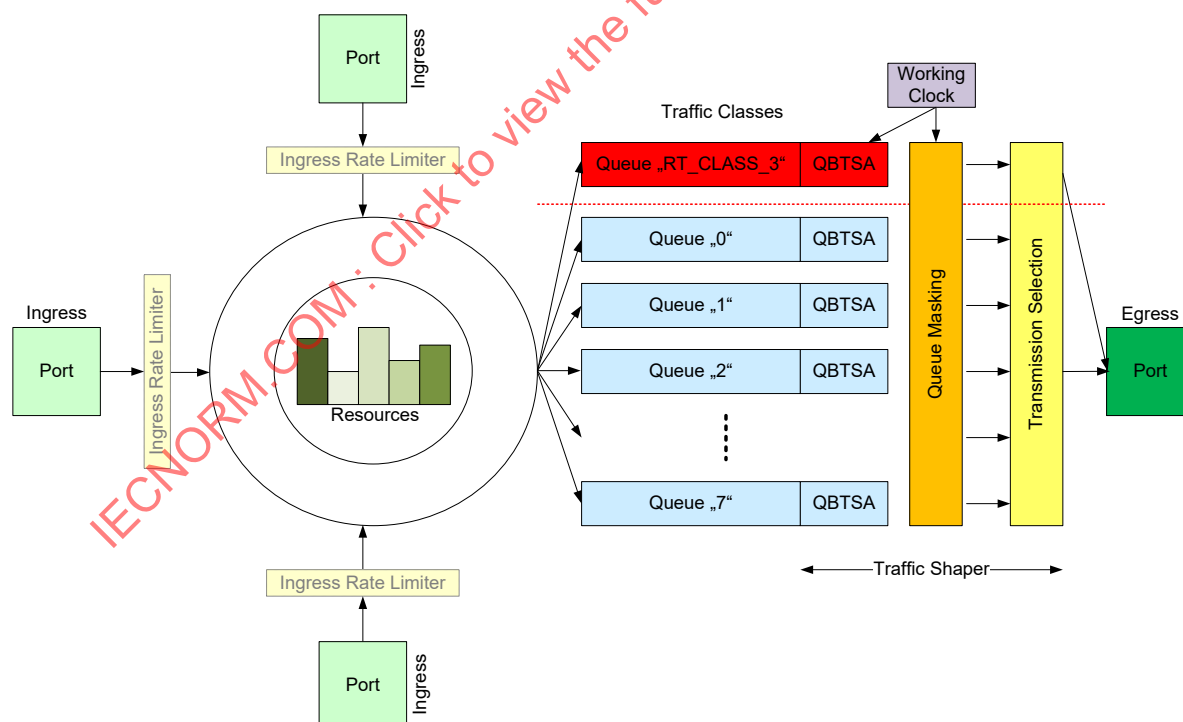


Figure 86 – Principle traffic flow model of a bridge

4.12.2.1.11.1.3 Resources

The bridge shall provide and organize its resources in a way to ensure robustness for the traffic defined in this document as shown in Figure 87 and Table 324.

The frame resources for the queues are allocated from a global “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 are blocked without frame dropping due to the used traffic shaping.

Table 324 – Queue related memory management

Queue	Comment
0	Global and dedicated resource pool
1	Global and dedicated resource pool
2	Global and dedicated resource pool
3	Global and dedicated resource pool
4	Global and dedicated resource pool
5	Global and dedicated resource pool
6	Global and dedicated resource pool
7	Global and dedicated resource pool

The Current Fill Level of a dedicated resource pool would offer information for predictive maintenance.

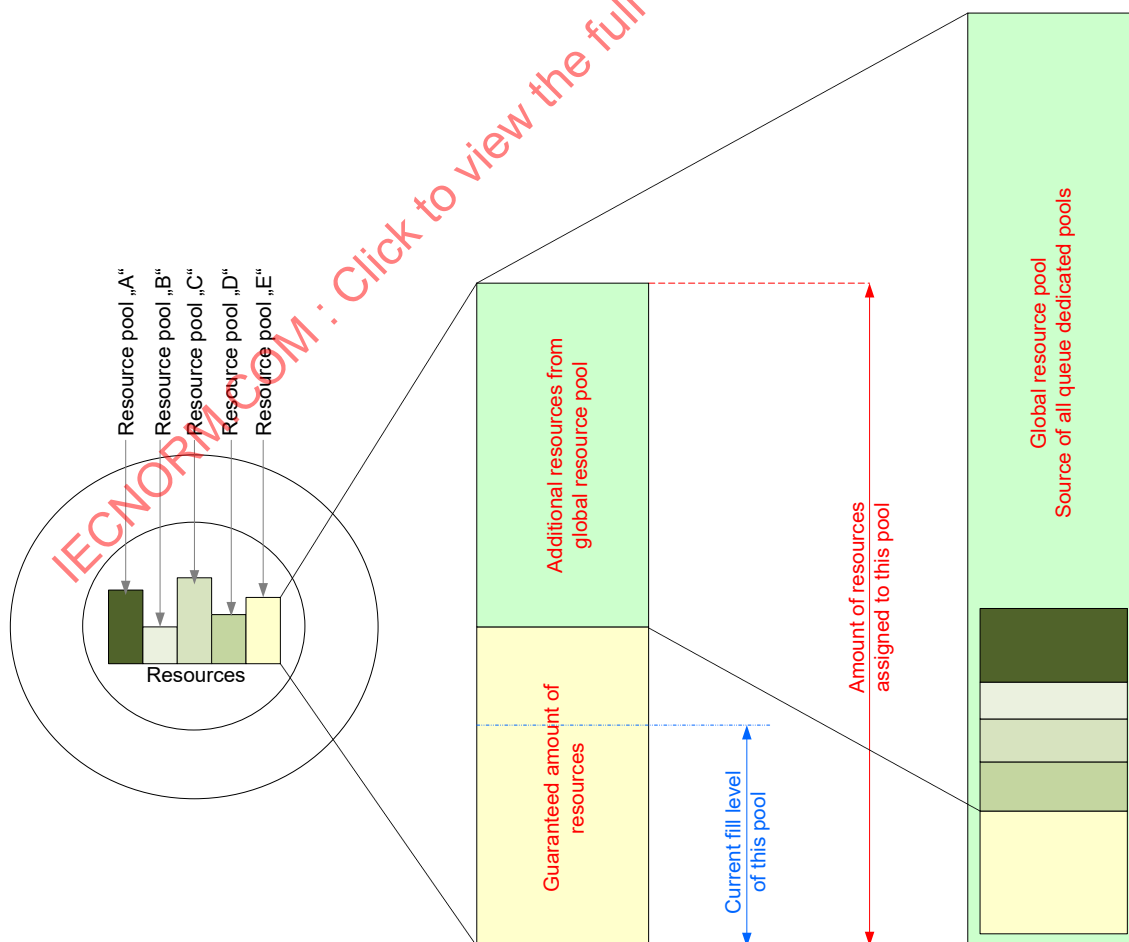


Figure 87 – Principle resource model of a bridge

4.12.2.1.11.2 Non time aware bridge

4.12.2.1.11.2.1 Without RT_CLASS_3

Identify the traffic class and thus, the queue in which the frame will be stored. Table 325 shows the queue usage.

Table 325 – Queue usage

Queue	RED_RELAY	Fragmentation	Cut through	Comment
0	—	Yes	Yes	—
1	—	Yes	Yes	—
2	—	Yes	Yes	Frames used for connection establishment, records and control
3	—	Yes	Yes	—
4	—	Yes	Yes	—
5	—	Yes	Yes	Acyclic transmitted frames rated by this document as RTA_CLASS_x / Alarm
6	—	Yes	Yes	Cyclic transmitted frames rated by this document as RT_CLASS_1, 2, UDP
7	—	No	Yes	Synchronization and Network Management Non preemptable protocols

4.12.2.1.11.2.2 With RT_CLASS_3

Identify the traffic class and thus, the queue in which the frame will be stored. Table 326 shows the queue usage.

Table 326 – Queue usage

Queue	RED_RELAY	Fragmentation	Cut through	Comment
0	—	Yes	Yes	—
1	—	Yes	Yes	—
2	—	Yes	Yes	Frames used for connection establishment, records and control
3	—	Yes	Yes	—
4	Yes	No	Yes	Cyclic transmitted frames rated by this document as RT_CLASS_3
5	—	Yes	Yes	Acyclic transmitted frames rated by this document as RTA_CLASS_x / Alarm
6	—	Yes	Yes	Cyclic transmitted frames rated by this document as RT_CLASS_1, 2, UDP
7	—	No	Yes	Synchronization and Network Management Non preemptable protocols

4.12.2.1.12 Queue management

See IEEE 802.1Q chapter 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
- to ensure overall QoS

4.12.2.1.13 Queue based transmission selection

4.12.2.1.13.1 General

See IEEE 802.1Q chapter 8.6.7.

This module provides per queue a “frame available” flag which is used by “Normal transmission selection” and “Express transmission selection”.

This flag is created per queue using

- strict priority QBTSA, which signals “frame available” if queue contains a frame,
- weighted round robin QBTSA, which signals “frame available” if queue contains a frame and assigned bandwidth is still available.

4.12.2.1.13.2 Non time aware bridge

4.12.2.1.13.2.1 Without RT_CLASS_3

Table 327 shows the assigned queue based transmission selection.

Table 327 – QBTSA usage

Queue	QBTSA	Comment
0	strict priority or weighted round robin	Normal transmission selection
1	strict priority or weighted round robin	Normal transmission selection
2	strict priority or weighted round robin	Normal transmission selection
3	strict priority or weighted round robin	Normal transmission selection
4	strict priority or weighted round robin	Normal transmission selection
5	strict priority	Normal transmission selection
6	strict priority	Normal transmission selection
7	strict priority	Normal transmission selection

4.12.2.1.13.2.2 With RT_CLASS_3

Table 328 shows the assigned queue based transmission selection.

The QBTSA “Fragmentation” is defined by this document. It uses the WorkingClock and the Phase Control to segment frames and influences the Queue Masking outside of RED PHASE.

Table 328 – QBTSA usage

Queue	QBTSA	Comment
0	strict priority or weighed round robin or Fragmentation	Normal transmission selection
1	strict priority or weighed round robin or Fragmentation	Normal transmission selection
2	strict priority or weighed round robin or Fragmentation	Normal transmission selection
3	strict priority or weighed round robin or Fragmentation	Normal transmission selection
4	strict priority	Used for RT_CLASS_3 Normal transmission selection
5	strict priority or Fragmentation	Normal transmission selection
6	strict priority or Fragmentation	Normal transmission selection
7	strict priority	Normal transmission selection

4.12.2.1.14 Queue masking**4.12.2.1.14.1 General**

See IEEE 802.1Q.

The “Queue based transmission selection” provides per queue a “frame available” flag which may be overwritten by time aware shaper and pre-emption.

The WorkingClock is used as time information if time is used as masking criteria.

With Gate Open for one queue only this queue is visible for the transmission selection.

4.12.2.1.14.2 Non time aware bridge**4.12.2.1.14.2.1 Without RT_CLASS_3**

Used as specified by IEEE 802.1Q.

4.12.2.1.14.2.2 With RT_CLASS_3

The queue for RT_CLASS_3 is handled similar 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 RT_CLASS_3 queue is visible for the transmission selection.

As soon as the RED PHASE ends, the RT_CLASS_3 queue is invisible for the transmission selection.

Outside of the RED PHASE, used as specified by IEEE 802.1Q

4.12.2.1.15 Normal transmission selection

See IEEE 802.1Q chapter 8.6.8.

This module fetches a frame from the queue with the highest priority traffic class and “frame available” flag equals true. This process is repeated after each frame.

4.12.2.1.16 Express transmission selection

See IEEE 802.1Q chapter 8.6.8.

This module fetches a frame from the pre-emptive queue if “frame available” flag equals true. This process is repeated after each frame.

4.12.2.1.17 Normal MAC

See IEEE 802.1Q and IEEE 802.3.

Transmission or reception of a frame from all queues.

4.12.2.1.18 Express MAC

See IEEE 802.1Q and IEEE 802.3.

Transmission or reception of a frame from the pre-emptive queues.

4.12.2.2 One port bridge end station

Figure 88 and Figure 89 show the application model of a one port bridge end station according to IEEE 802.1Q.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019

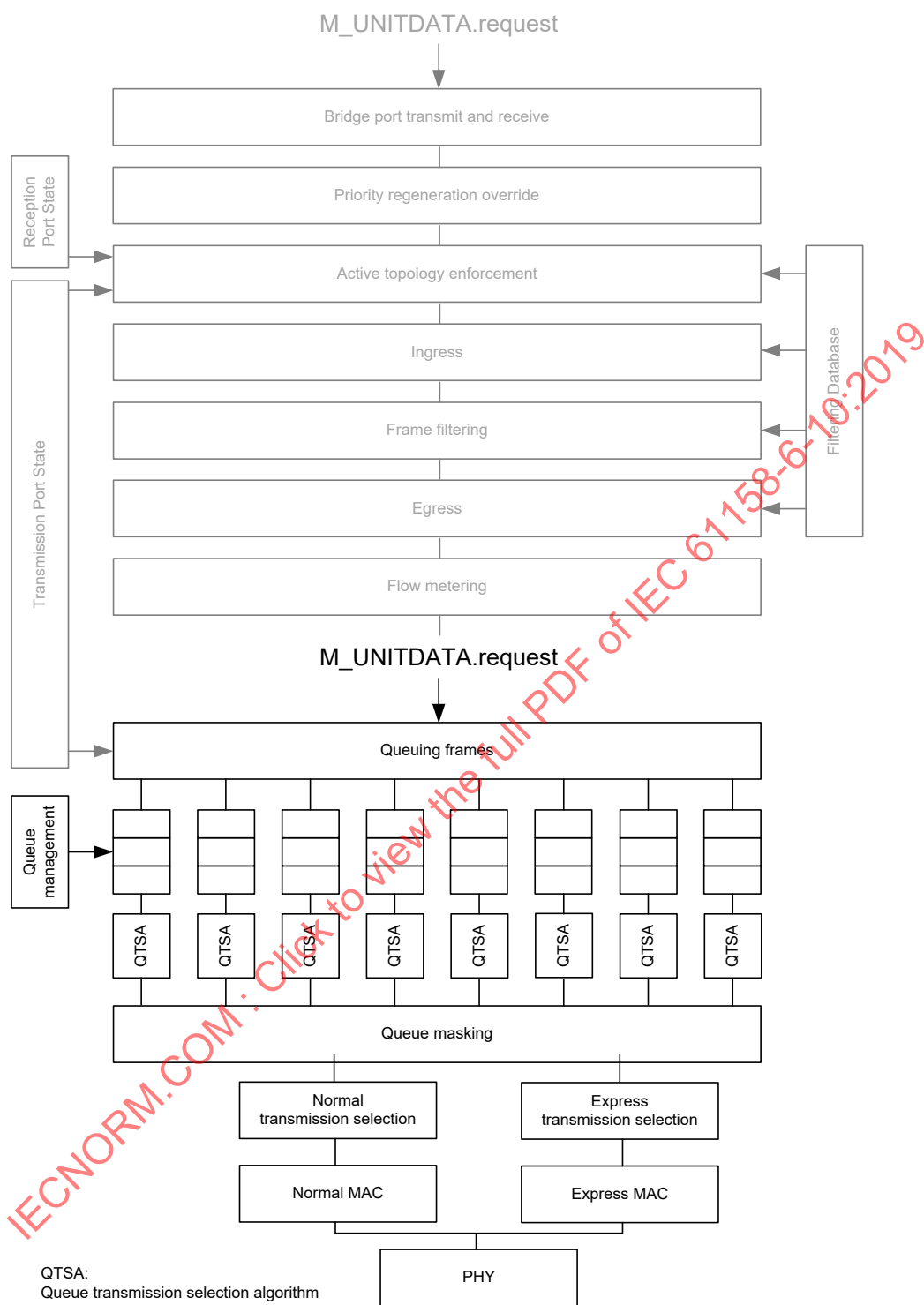


Figure 88 – End station – on port bridge – transmit

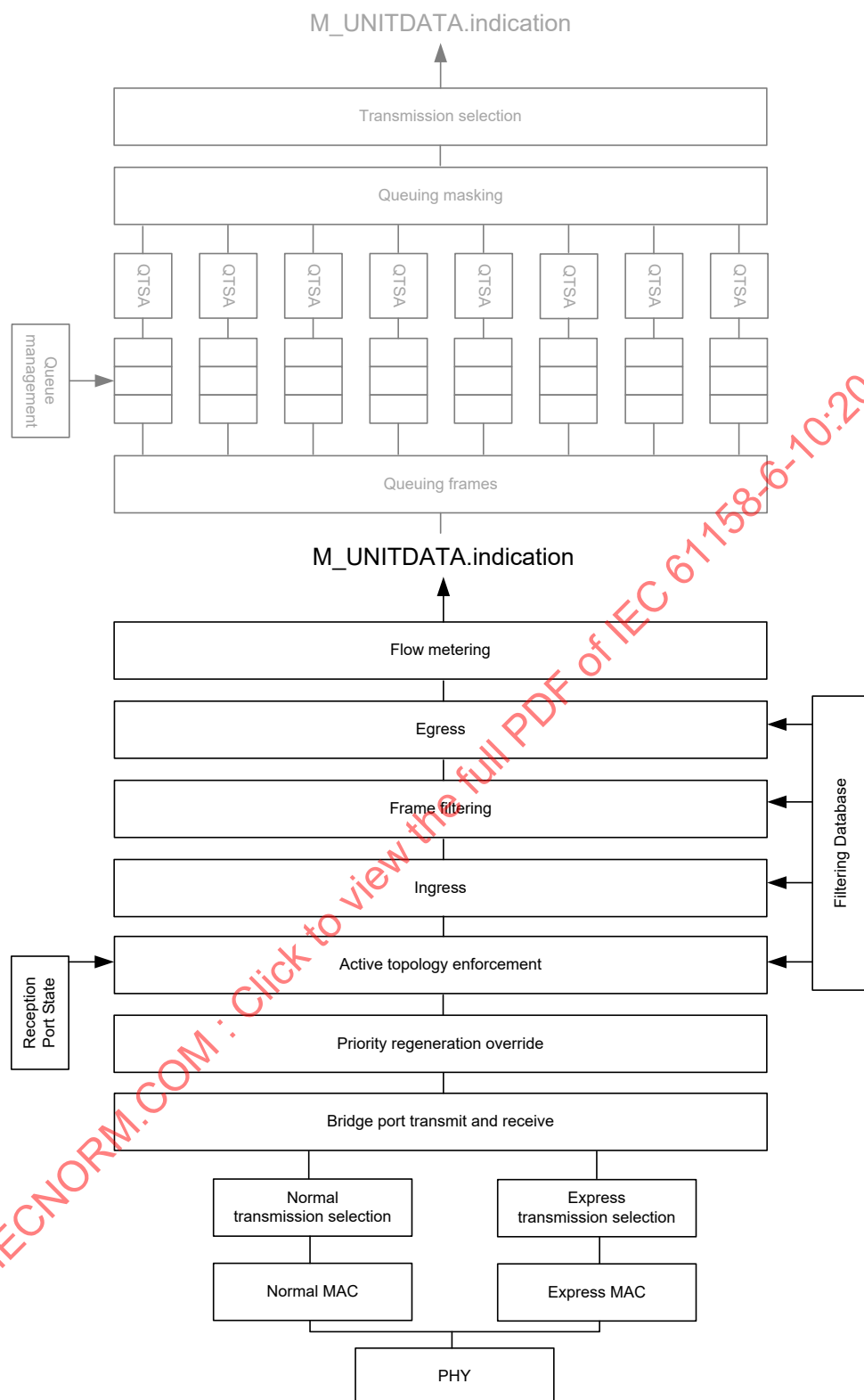


Figure 89 – End station – on port bridge – receive

4.12.2.3 Two port bridge with end station

Figure 90, Figure 91, Figure 92, Figure 93, Figure 94, and Figure 95 show the application model of an two port bridge with a management port hosting an end station according to IEEE 802.1Q.

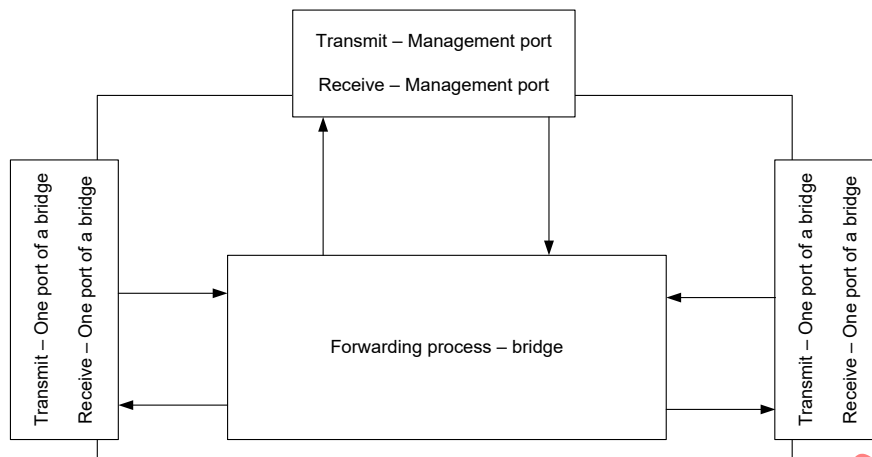


Figure 90 – Bridge with End Station

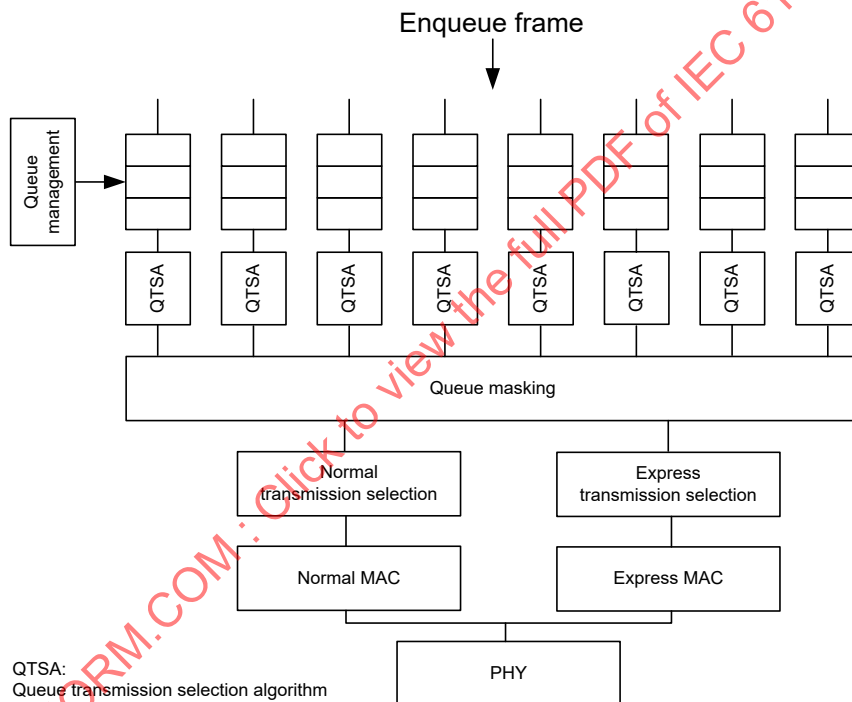


Figure 91 – Transmit – one port of a bridge

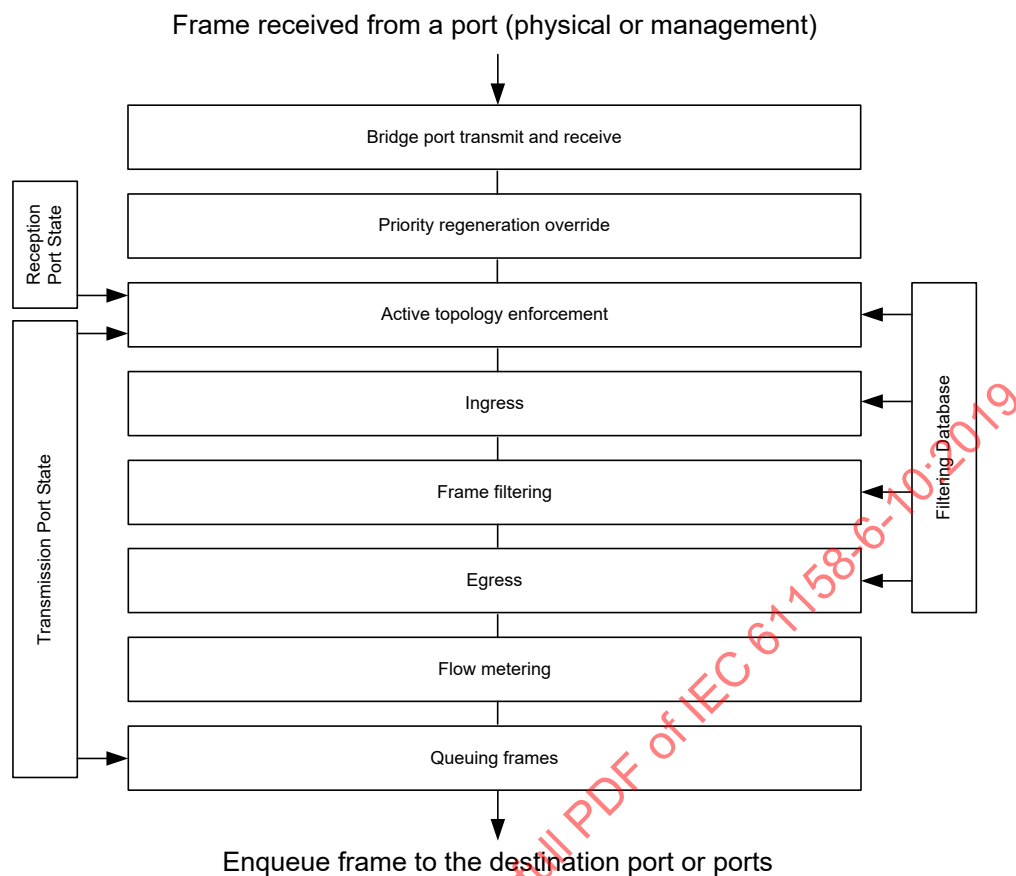


Figure 92 – Forwarding process – bridge

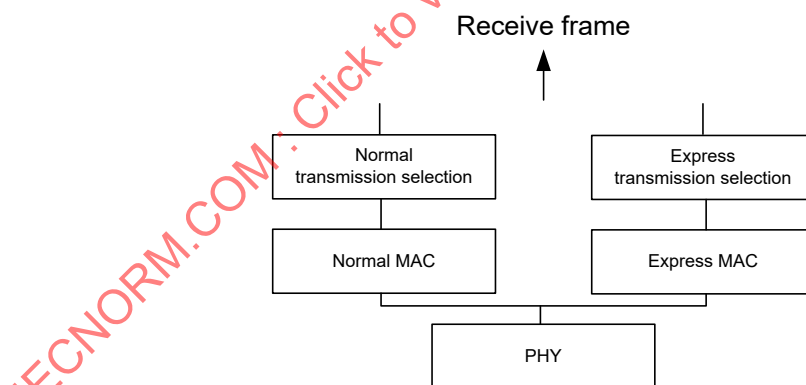


Figure 93 – Receive – on port of a bridge

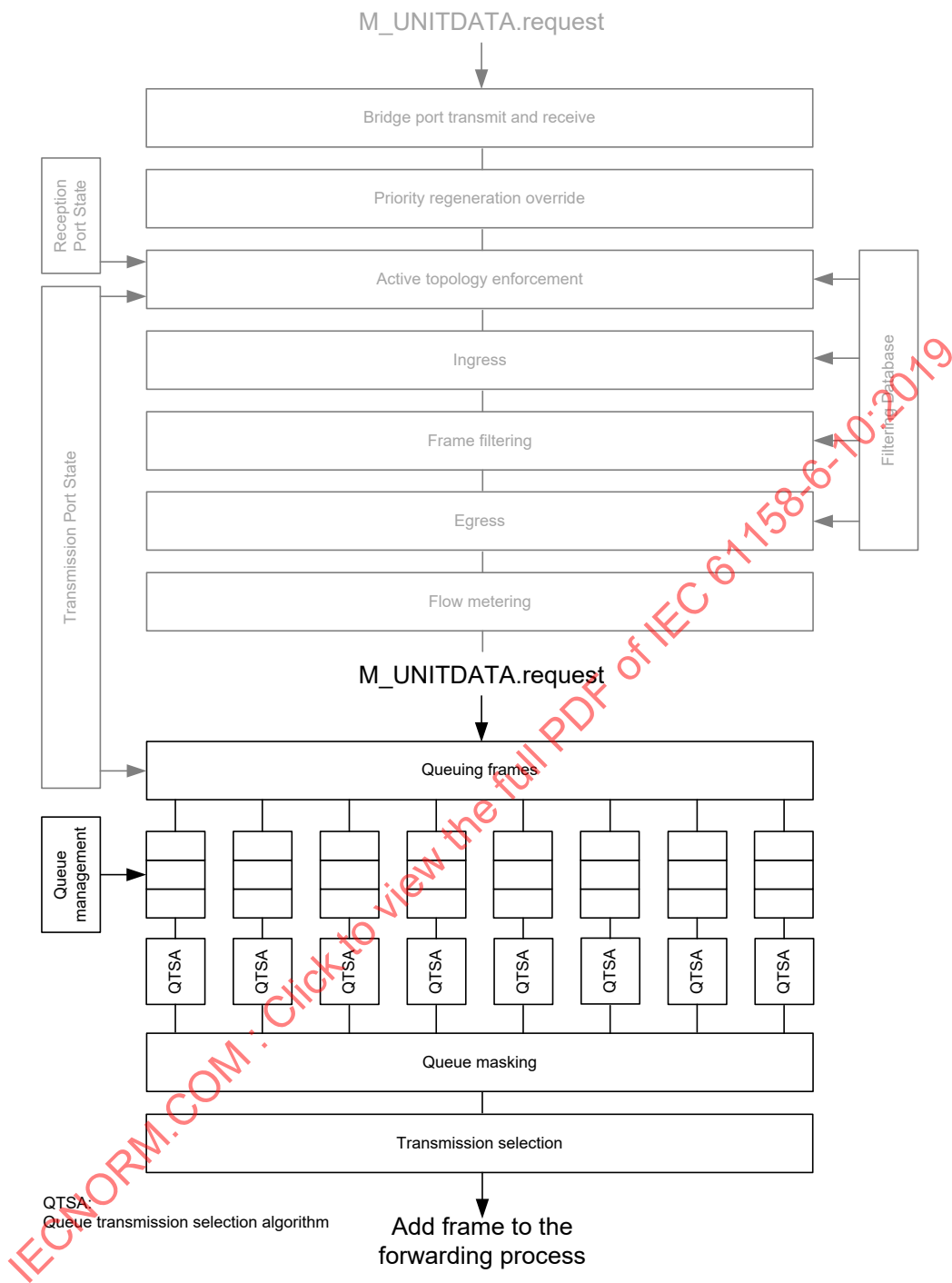


Figure 94 – Transmit – Management port

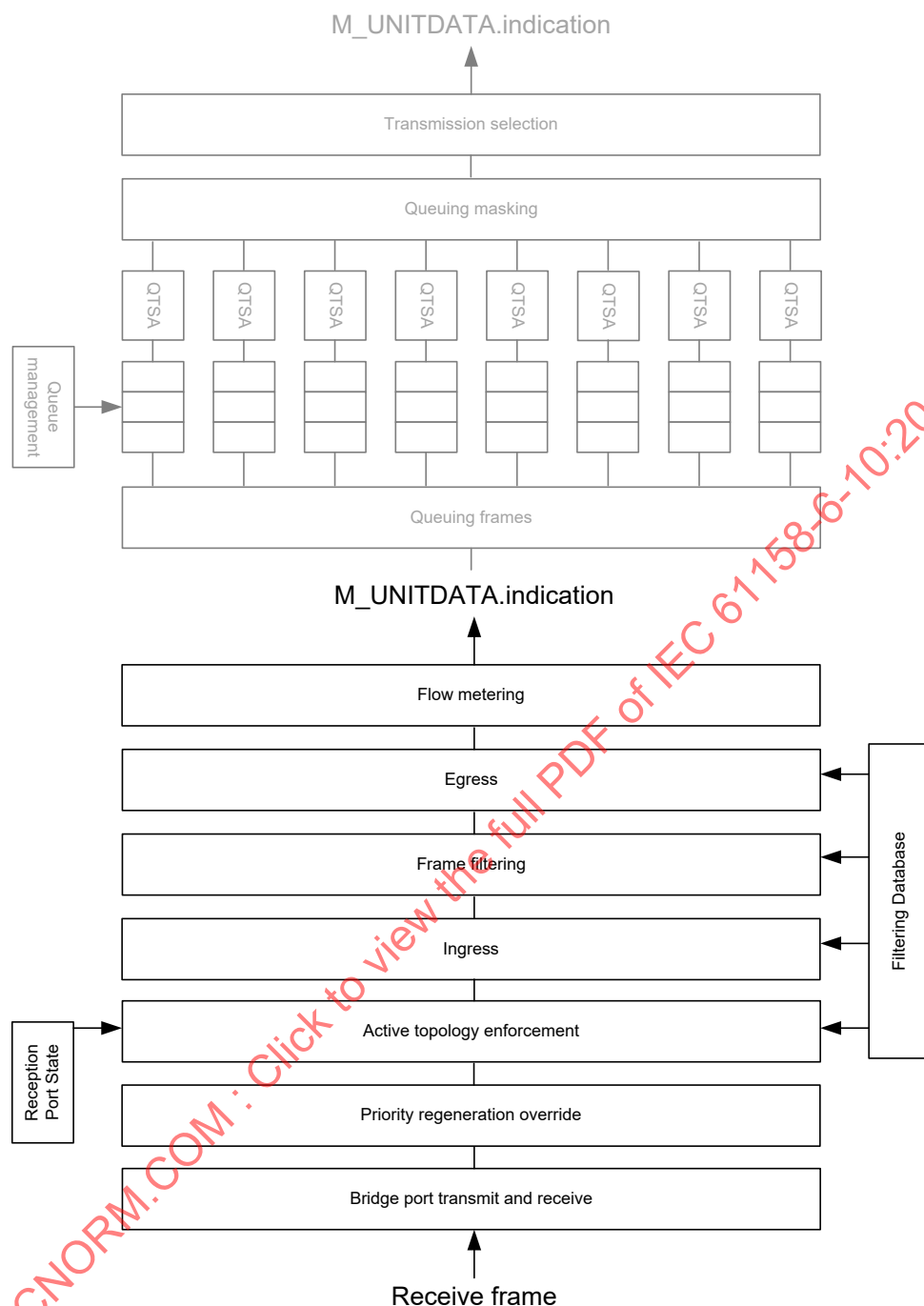


Figure 95 – Receive – Management port

4.12.2.4 n port bridge with end station

Same as 4.12.2.3 but with more ports.

4.12.3 Traffic Shaping

4.12.3.1 Traffic Classes and Traffic Shaper

The bridge shall organize its traffic classes and select a fitting traffic shaper in a way to ensure latency for the traffic defined in this document as shown in Table 329 and Table 330.

The traffic shaper shall use strict priority. If RT_CLASS_3 traffic is activated, then the definition of the RED_RELAY applies for the traffic shaper.

Table 329 – Traffic Classes[0..7] for eight queues

TCI.PCP	Traffic Class [Queue]	Comment
No TCI tag	0	Frames without VLAN tag and one to one assignment
0		Highest latency
1	1	One to one assignment
2	2	One to one assignment
3	3	One to one assignment
4	4	One to one assignment
5	5	One to one assignment
6	6	One to one assignment
7	7	Defined EtherType which requires this traffic class and one to one assignment
EtherType ^a		Lowest latency

^a Network Management or other well defined EtherTypes, for example Loop prevention or Synchronization

Table 330 – Traffic Classes[0..3] for four queues

TCI.PCP	Traffic Class [Queue]	Comment
No TCI tag	0	Highest latency
0		
1		
2	1	One to one assignment
3		
4	2	One to one assignment
5		
6	3	Lowest latency
7		
EtherType ^a		
^a Network Management or other well defined EtherTypes, for example Loop prevention or Synchronization		

4.12.4 Bridge extensions

Apart from IEEE 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 802.1Q shall be disabled according to Table 331, Table 332 and Table 333. As an optimization a previous learned entry of the FDB may be refreshed.

The routing mechanisms of IEEE 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.

4.12.5 QueueHandler

The QueueHandler manages the priority queues of a port. It offers service primitives for enqueueing and dequeuing of frames. Additionally, information of the content of a queue is available.

4.12.6 FAL Service Protocol Machines

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

4.12.7 Application Relation Protocol Machines

There is no Application Relation Protocol Machine (ARPM) defined for this Protocol.

4.12.8 DLL Mapping Protocol Machines

4.12.8.1 Forwarding Protocol Machine

4.12.8.1.1 Filtering database

According to IEEE 802 and this document a switch contains a filtering data base (FDB) to optimize the bridging. Table 331, Table 332 and Table 333 show the list of entries.

The row ORG with an entry “YES” indicates whether this protocol is able to be received or transmitted, even if a port is excluded from the active topology by MRP or RSTP.

NOTE IEC 62439-2 if supported adds further entries in the FDB.

Table 331 – Unicast FDB entries

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Applicati on	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 332 – Multicast FDB entries

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Application	Bridge	ORG	Learning	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	00-01-01	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
01-0E-CF	00-04-20	Receive ^a	Filter	Yes	Disabled	In conjunction with PTCP-RTSyncPDU with follow up and FrameID(=0x0020) used for working clock synchronization
01-0E-CF	00-04-40	Receive ^a	Filter	Yes	Disabled	In conjunction with PTCP-FollowUpPDU and FrameID(=0xFF20) used for working clock synchronization
01-0E-CF	00-04-80	Receive ^a	Filter	Yes	Disabled	In conjunction with PTCP-RTSyncPDU and FrameID(=0x0080) used for working clock synchronization
01-0E-CF	00-05-00	Receive ^a	Forward ^a	Yes	Disabled	Additional managed multicast MAC address used vendor specific in conjunction with MRP.
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	Yes	Disabled	IEEE 802.1Q Rapid spanning tree protocol (RSTP)
01-80-C2	00-00-01 – 00-00-0D	Receive ^a	Filter	Yes	Disabled	IEEE 802.1Q
01-80-C2	00-00-0E	Receive ^a	Filter	Yes	Disabled	IEEE 802.1Q, IEEE 802.1AS, IEEE 802.1AB (LLDP), and 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	Filter	Yes	Disabled	IEEE 802.1Q
01-80-C2	00-00-10	Receive ^a	Filter	Yes	Disabled	IEEE 802.1Q All LANs bridge management group address

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Applicati on	Bridge	ORG	Learn- ing	Meaning
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 This frames shall be discarded by the MAC Relay if RT_CLASS_3 is supported, otherwise 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.						

Table 333 – Broadcast FDB entry

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Applicati on	Bridge	ORG	Learn- ing	Meaning
FF-FF-FF	FF-FF-FF	Receive ^a	Forward	No	Enabled	Broadcast
Others	Others	—	—	—	—	—
^a If applicable.						

4.12.8.1.2 Primitive definitions

4.12.8.1.2.1 General

The service primitives including their associated parameters issued by MAC_RELAY user received by MAC_RELAY and vice versa are described in the MAC bridges ASE in the service definition.

4.12.8.1.2.2 Primitives exchanged between remote machines

The service primitives including their associated parameters issued or received by MAC_RELAY are described in the service definition and shown in Table 334.

Table 334 – 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	—

4.12.8.1.2.3 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 335.

Table 335 – Local primitives issued or received by MAC_RELAY

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.12.8.1.3 State transition diagram

The state transition diagram of the MAC_RELAY shall be according to IEEE 802.1Q.

4.12.8.1.4 State machine description

The state machine shall of the MAC_RELAY be according to IEEE 802.1Q.

4.12.8.1.5 MAC_RELAY state table

The state table shall be according to IEEE 802.1Q.

4.12.8.1.6 Functions, Macros, Timers and Variables

Table 336 contains the functions, macros, timers and variables used by the MAC_RELAY and their arguments and their descriptions.

Table 336 – Functions, Macros, Timers and Variables used by the MAC_RELAY

Name	Type	Function/Meaning
—	—	—

4.12.8.2 RT_CLASS_3 port state machine

4.12.8.2.1 General

This part of the specification defines the utilization of the IEEE 802.1Q.

It includes extensions for RT_CLASS_3 phase regarding forwarding of frames.

The routing mechanisms of IEEE 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.

4.12.8.2.2 Primitive definitions

4.12.8.2.2.1 Primitives exchanged between remote machines

The 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 337.

Table 337 – Remote primitives issued or received by RTC3PSM

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.12.8.2.2 Primitives exchanged between local machines

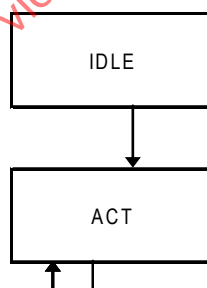
The local service primitives including their associated parameters issued or received by RTC3PSM are described in the service definition and shown in Table 338.

Table 338 – 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 802.3	RTC3PSM	Local Port ID, Local MAU Type, Local Link Status	—
Port_State_Change_ind	IEEE 802.1Q	RTC3PSM	Local PortState	—
SyncStateChange_ind	PTCP	RTC3PSM	Sync State	—
SetPortState	RTC3PSM	IEEE 802.1Q	Local PortState	—

4.12.8.2.3 State transition diagram

The state transition diagram of the RTC3PSM is shown in Figure 96.

**Figure 96 – State transition diagram of RTC3PSM**

States of the RT3PSM are:

IDLE	Idle state
ACT	Active state

4.12.8.2.4 State machine description

The Real-time Class 3 Port State Machine (RTC3PSM) arranges the port states of real-time 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:

A device, only loosely coupled due to missing hardware support for SYNC, shall be supported. In this case, it is indicated by an internally defined LineDelay:=“DummyMode”,

DIFFERENCE_FAULT(CableDelay) and the check of the LineDelay shall not return a fault for this port.

4.12.8.2.5 RTC3PSM state table

The RTC3PSM state table is shown in Table 339 and shall exist for each port. The bandwidth allocation using the TXBeginEndAssignment and the RXBeginEndAssignment shall be coupled with the RTC3PSM port states.

Table 339 – 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

4.12.8.2.6 Functions, Macros, Timers and Variables

Table 340 contains the functions, macros, timers and variables used by the RTC3PSM and their arguments and their descriptions.

Table 340 – 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 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 local stored information which is expected to be achieved from remote. The expected values are extracted from the parametrization (for example PDPorDataCheck record) which shall contain all needed data.
EXPECTED_REMOTE	Macro	This function delivers the local stored information which is expected to be delivered from remote. The expected values are extracted from the parametrization (for example PDPorDataCheck 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 duplexity is full according to 6.2.12.7.
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 duplexity is full according to Table 638.
REMOTE	Macro	This function delivers the remote information stored locally.

Name	Type	Function/Meaning
REMOTEPORTSTATE	Macro	(// physical Does the remote RT_CLASS_3_Port_State exist and contain a value out of 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.

4.12.8.2.7 Truth table

Table 341 and Table 342 contain the truth table used by the RTC3PSM.

Table 341 – 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 is valid.

^c RT_CLASS_3 frames should only be transmitted in this case.

Table 342 – 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.

4.12.8.2.8 Generating of events

The RTC3PSM shows the state changes required to establish a RT_CLASS_3 peer to peer communication. Additionally, Figure 97 shows the diagnostic events, generated according to the state changes of the RTC3PSM. Table 343 defines the coding of the associated events.

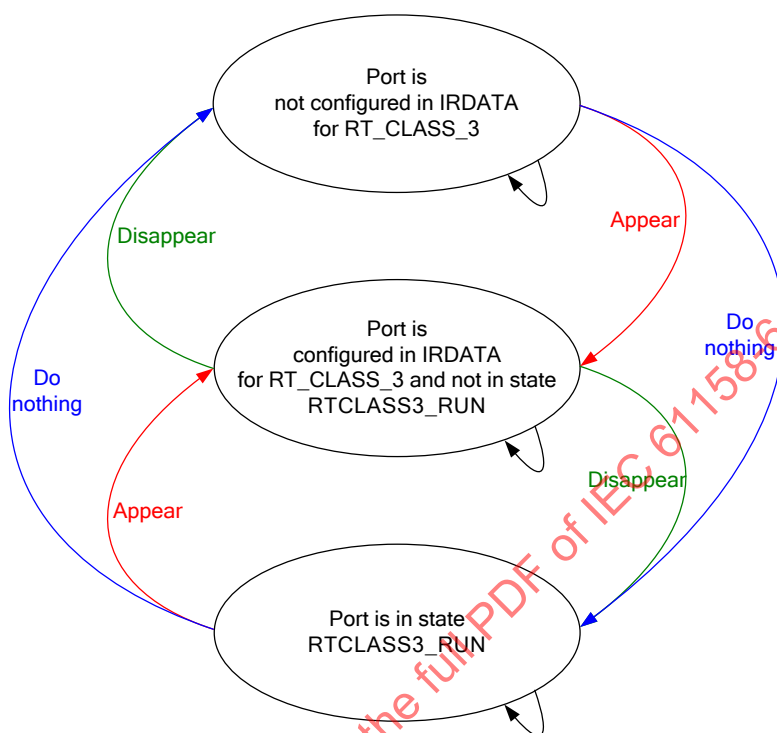


Figure 97 – State transition diagram for generating events

Table 343 – 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 show the principle. The rules for ChannelProperties.Specifier apply.

4.12.8.3 RT_CLASS_3 Forwarding Protocol Machine

4.12.8.3.1 Primitive definitions

4.12.8.3.1.1 Primitives exchanged between remote machines

The 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 344.

Table 344 – Remote primitives issued or received by RED_RELAY

Primitive	Source	Destination	Associated parameters	Functions
DMUX_RED_RELAY.ind	DMP_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.

4.12.8.3.1.2 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 345.

Table 345 – 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	—

4.12.8.3.2 State transition diagram

The state transition diagram of the RED_RELAY is shown in Figure 98.

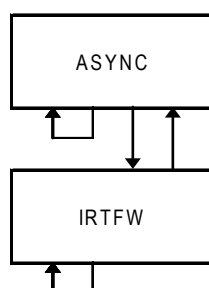


Figure 98 – State transition diagram of RED_RELAY

States of the RED_RELAY are:

ASYNC	The node shall be synchronized, either local as a sync master or remote as a sync slave for the RED forwarding
IRTFW	If the node is synchronized, the forwarding and local reception for RED frames using the PDIRFrameData or the PDIRGlobalData and PDIRBeginEndData shall be done

4.12.8.3.3 State machine description

The Realtime Class 3 Forwarding Protocol Machine (RED_RELAY) arranges the forwarding and local reception of RED frames.

4.12.8.3.4 RED_RELAY state table

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 43 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 199 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 out of the PDIRGlobalData.

The RED_RELAY state table is shown in Table 346.

Table 346 – RED_RELAY state table

#	Current State	Event /Condition =>Action	Next State
1	ASYN	SyncStateChange_ind () /SYNC => ignore	IRTFW
2	ASYN	SyncStateChange_ind () /!SYNC => ignore	ASYN
3	ASYN	TickEvent () => ignore	ASYN
4	ASYN	DMUX_RED_RELAY.ind () => ignore	ASYN
5	IRTFW	SyncStateChange_ind () /SYNC => ignore	IRTFW
6	IRTFW	SyncStateChange_ind () /!SYNC => ignore	ASYN
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

4.12.8.3.5 Functions, Macros, Timers and Variables

Table 347 contains the functions, macros, timers and variables used by the RED_RELAY and their arguments and their descriptions.

Table 347 – 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	Calculate the input parameter Local_FrameSendOffset for the RedGuard
Calculate Local_Phase as input for the RedGuard	Macro	Calculate the input parameter Local_Phase for the RedGuard
IRT_Timer	Timer	Timer which is used to controls 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

4.12.8.3.6 RedGuard

The RedGuard checks the received frame (which is “ReceivedInRED == TRUE” due to the DEMUX) against Table 348, Table 349 or Table 350. The used table depends on the local supported functionality.

Table 348 – 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 could be done keeping its given FrameSendOffset (TRUE) or not (FALSE).

Table 349 – 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 could be done keeping its given FrameSendOffset (TRUE) or not (FALSE).				

Table 350 – 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
FALSE		—	—
TRUE		—	! RUN
TRUE		Yes	RUN
^a The FrameID shall be checked against PDIRFrameData (for local received and not forwarded frames) and PDIRBeginEndData. ^b The bridge should discard the frame.			

4.12.8.4 DFP Forwarding Protocol Machine

4.12.8.4.1 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 99, Figure 100 and Figure 101 show the integration of DFP.

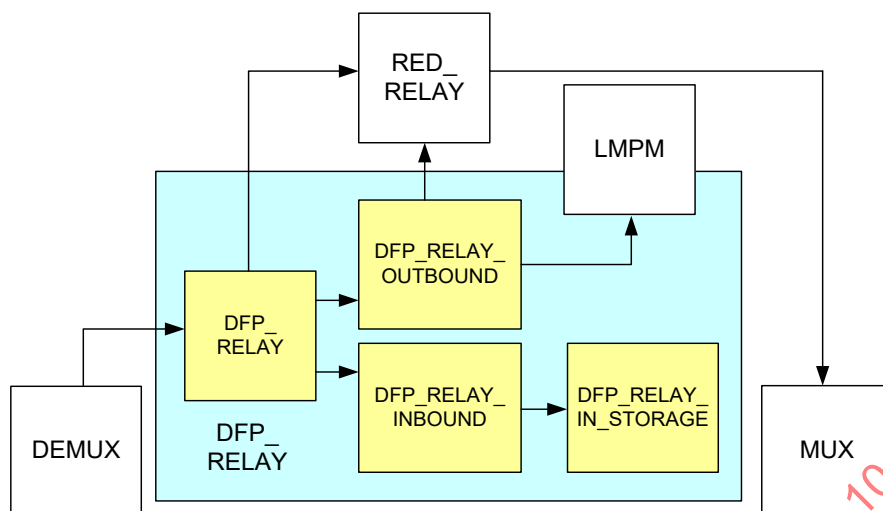


Figure 99 – Scheme of the DFP_RELAY

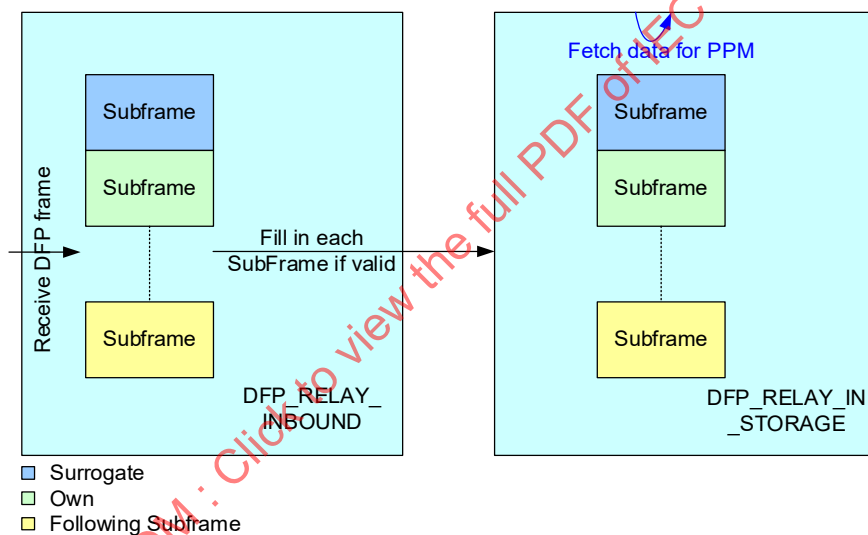


Figure 100 – Scheme of the DFP_RELAY_INBOUND and DFP_RELAY_IN_STORAGE

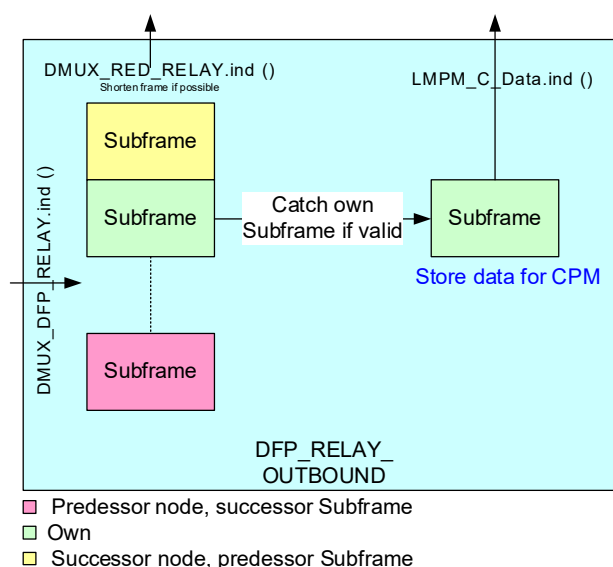


Figure 101 – Scheme of the DFP_RELAY_OUTBOUND

4.12.8.4.2 Startup

The DFP_RELAY shall, like the RED_RELAY, start with the persistent stored data from earlier AR establishment, after power on.

Thus, dynamic frame packing works independently from the IO controller startup sequence as soon as the RED_RELAY is active.

4.12.8.4.3 DFP Protocol Machine

4.12.8.4.3.1 Primitive definitions

4.12.8.4.3.1.1 Primitives exchanged between remote machines

The 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 351.

Table 351 – 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	—

4.12.8.4.3.1.2 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 352.

Table 352 – Local primitives issued or received by DFP_RELAY

Primitive	Source	Destination	Associated parameters	Functions
SyncStateChange_ind	PTCP	DFP_RELAY	state {SYNC, ASYNC}	

4.12.8.4.3.2 State transition diagram

The state transition diagram of the DFP_RELAY is shown in Figure 102.

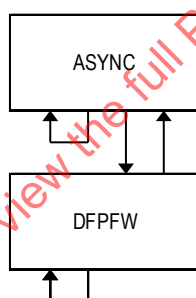


Figure 102 – State transition diagram of DFP_RELAY

States of the DFP_RELAY are:

ASYNC	The node shall be synchronized, either local as a sync master or remote as a sync slave for the DFP forwarding which is based on the RED forwarding
DFPFW	DFP forwarding active

4.12.8.4.3.3 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 not defined by PDIRSubframeData, are forwarded to the RED_RELAY and not handled by the DFP_RELAY.

4.12.8.4.3.4 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 353.

Table 353 – DFP_RELAY state table

#	Current State	Event /Condition =>Action	Next State
1	ASYN	SyncStateChange_ind () /SYNC => ignore	DFPFW
2	ASYN	SyncStateChange_ind () /!SYNC => ignore	ASYN
3	DFPFW	SyncStateChange_ind () /SYNC => ignore	DFPFW
4	DFPFW	SyncStateChange_ind () /!SYNC => ignore	ASYN
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

4.12.8.4.3.5 Functions, Macros, Timers and Variables

Table 354 contains the functions, macros, timers and variables used by the DFP_RELAY and their arguments and their descriptions.

Table 354 – 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.

4.12.8.4.3.6 DFPGuard

The DFPGuard checks the received frame against Table 355.

Table 355 – 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's not part of the database, then forward the frame to the RED_RELAY
TRUE	TRUE	FALSE	FALSE	TRUE	FALSE	The local DFP database return the frame attribute "OUTBOUND" for the given FrameID
TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	The local DFP database return 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.

4.12.8.4.4 DFP Inbound Protocol Machine

4.12.8.4.4.1 Primitive definitions

4.12.8.4.4.1.1 Primitives exchanged between remote machines

The 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 356.

Table 356 – 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	—

4.12.8.4.4.1.2 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 357.

Table 357 – 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.

4.12.8.4.4.2 State transition diagram

The state transition diagram of the DFP_RELAY_INBOUND is shown in Figure 103.

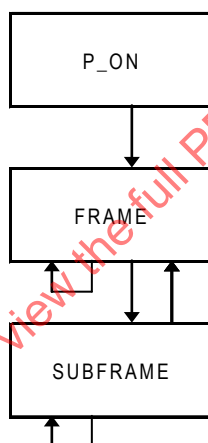


Figure 103 – State transition diagram of DFP_RELAY_INBOUND

States of the DFP_RELAY_INBOUND are:

P_ON	Data initialization
FRAME	Check DFP frame
SUBFRAME	Check DFP sub frame

4.12.8.4.4.3 State machine description

This state machine receives the INBOUND DFP frames, identified by it's FrameID, checks them against the protocol definition supporting the two modes DistributedWatchDog active and passive.

For MRPD, two DFP layouts exist, identified by the FrameID.

4.12.8.4.4.4 DFP_RELAY_INBOUND state table

The DFP_RELAY_INBOUND shall ensure that the reception of a DFP frames is done in a correct way and the state table is shown in Table 358.

Table 358 – 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

4.12.8.4.4.5 Functions, Macros, Timers and Variables

Table 359 contains the functions, macros, timers and variables used by the DFP_RELAY_INBOUND and their arguments and their descriptions.

Table 359 – 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 – sub frame data check returns Invalid.
SubframeDataValid	Macro	The InboundGuard – sub frame data check returns Valid.
SubframeStructureInvalid	Macro	The InboundGuard – sub frame check returns Invalid.
SubframeStructureValid	Macro	The InboundGuard – sub frame check returns Valid.

4.12.8.4.4.6 InboundGuard

The InboundGuard checks the received frame against Table 360, Table 361, Table 362 and Table 363.

Table 360 – Truth table for the InboundGuard – frame check

Input	Output
Header Subframe. SFCRC16	Frame check
Invalid	Invalid
Valid	Valid

Table 361 – Truth table for the InboundGuard – sub frame check

Input			Output
Subframe. Position ^a	Subframe. Length ^b	Subframe. SFCRC16 ^c	Sub frame 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 362 – Truth table for the InboundGuard – sub frame data check

Input				Output
RestartFor-DistributedWdTime ^a	Distributed-WatchDog-Factor ^b	Subframe. DataStatus. Valid ^c	Subframe. SFCycleCounter ^d	Sub frame 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 RestartForDistributedWdTime 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 363 – 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

4.12.8.4.5 DFP Inbound Storage Protocol Machine

4.12.8.4.5.1 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.

4.12.8.4.5.2 Primitive definitions

4.12.8.4.5.2.1 Primitives exchanged between remote machines

The 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 364.

Table 364 – 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, ModuleIdentifier, SubmoduleIdentifier, AlarmItem	Inform the application that a concatenation fault takes place.

4.12.8.4.5.2.2 Primitives exchanged between local machines

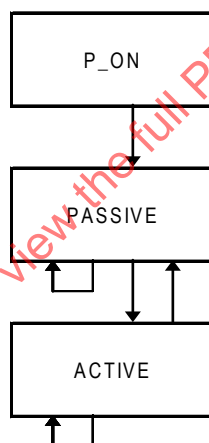
The local service primitives including their associated parameters issued or received by DFP_RELAY_IN_STORAGE are described in the service definition and shown in Table 365.

Table 365 – 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 on its storage

4.12.8.4.5.3 State transition diagram

The state transition diagram of the DFP_RELAY_IN_STORAGE is shown in Figure 104.

**Figure 104 – 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 sub frame with distributed watchdog if activated. Additional check the concatenation of the subframes against the associated PPM time constraints and generate indications if necessary.

4.12.8.4.5.4 DFP_RELAY_IN_STORAGE state machine description

This state machine receives subframe and stores them for further use. The concatenation of the subframes to the local 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 802.3. In this case the last stored data of the subframe is still valid and useable.

4.12.8.4.5.5 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 366.

Table 366 – 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
12	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
		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 ()	
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 show 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

4.12.8.4.5.6 Functions, Macros, Timers and Variables

Table 367 contains the functions, macros, timers and variables used by the DFP_RELAY_IN_STORAGE and their arguments and their descriptions.

Table 367 – 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 sub frame to zero and the DataStatus.Ignore of the sub frame 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	Local 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 updated the C_SDU, DataStatus and the SFCycleCounter of the sub frame
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 sub frame was received in time to be concatenated to the local injected frame.
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

Name	Type	Function/Meaning
RestartForDistributedWDTime	Timer	This timer is used to protect a surrogate sub frame for the detection of a fault by the IO controller. It is only active if the distributed watchdog is requested by the engineering.

4.12.8.4.6 DFP Outbound Protocol Machine

4.12.8.4.6.1 Primitive definitions

4.12.8.4.6.1.1 Primitives exchanged between remote machines

The 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 368.

Table 368 – Remote primitives issued or received by DFP_RELAY_OUTBOUND

Primitive	Source	Destination	Associated parameters	Functions
DFP_OUTBOUND.ind	DFP_RELAY_OUTBOUND	DFP_RELAY_OUTBOUND	Port, Tstamp, DA, SA, C_SDU, ReceivedInRED	—
LMPM_C_Data.ind	DFP_RELAY_OUTBOUND	—	CREP, DA, SA, VLANPri, VLANID, C_SDU, APDU_Status, ReceivedInRED	Convey the own sub frame 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	—

4.12.8.4.6.1.2 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 369.

Table 369 – 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	—

4.12.8.4.6.2 State transition diagram

The state transition diagram of the DFP_RELAY_OUTBOUND is shown in Figure 105.

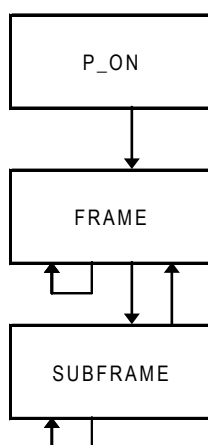


Figure 105 – State transition diagram of DFP_RELAY_OUTBOUND

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.

4.12.8.4.6.3 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 given to the LMPM and the, if applicable, shortened frame is given to the RED_RELAY.

The frame shall be shortened if SFIOCRProperties.SFCRC16 is set to one.

The shortening shall be done by replacing the locally received subframe with a SFEndDelimiter, if needed Padding and an APDU_Status according to Table 370.

Table 370 – 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 one.

4.12.8.4.6.4 DFP_RELAY_OUTBOUND state table

The DFP_OUTBOUND state table is shown in Table 371.

Table 371 – 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 shorten after the last valid subframe by the MUX according to the rules of IEEE 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
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 shorten after the last valid subframe by the MUX according to the rules of IEEE 802.3	FRAME
9	ANY	DFP_Close_req () => DFP_Close_cnf (+)	P_ON

4.12.8.4.6.5 Functions, Macros, Timers and Variables

Table 372 contains the functions, macros, timers and variables used by the DFP_RELAY_OUTBOUND and their arguments and their descriptions.

Table 372 – 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 – sub frame check returns the value Invalid for the Sub frame structure.
OWN	Variable	This local variable contains the information about the own subframe.

4.12.8.4.6.6 OutboundGuard

The OutboundGuard checks the received frame against Table 373 and Table 374.

Table 373 – 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 are only checked if the SFCRC16 shall exist.

^b The header SFCRC16 is check before the subframe checking starts.

^c The shortening is only possible if a consistent structure is found before the own subframe.

Table 374 – Truth table for the OutboundGuard – sub frame 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, optional against the structure from the PDIRSubframeData. ^b Check against the protocol definition, optional against the structure from the PDIRSubframeData.				

4.12.8.5 Network Access Sender protocol Machine

4.12.8.5.1 Primitive definitions

4.12.8.5.1.1 Primitives exchanged between remote machines

The service primitives including their associated parameters issued or received by Network Access Sender (MUX) are described in the service definition and shown in Table 375.

Table 375 – 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	—

4.12.8.5.1.2 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 376.

Table 376 – Local primitives issued or received by MUX

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.12.8.5.2 State transition diagram

The state transition diagram of the MUX is shown in Figure 106.

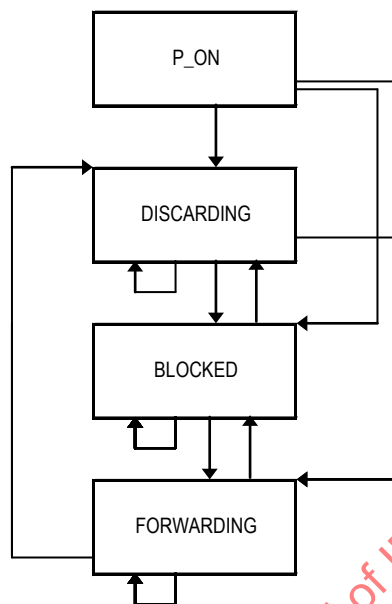


Figure 106 – State transition diagram of MUX

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

4.12.8.5.3 State machine description

The MUX is responsible for mapping LMPM, RED_RELAY and MAC_RELAY (IEEE 802.1Q) services to MAC according to IEEE 802.3.

Additionally, the Queue handler shown in Figure 109 is used to manage the priority queues.

4.12.8.5.4 MUX state table

The MUX state table is shown in Table 377.

Table 377 – 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
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

#	Current State	Event /Condition =>Action	Next State
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

4.12.8.5.5 Functions, Macros, Timers and Variables

Table 378 contains the functions, macros, timers and variables used by the MUX and their arguments and their descriptions.

Table 378 – 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 802.1Q egress filtering
FrameSizeFits	Function	This local function checks if the found frame fits into the remaining period till the next StartOfRED. It was only searched in the (...) queues from highest to lowest priority. The first non fitting frame stops the search and FALSE was 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

Name	Type	Function/Meaning
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

4.12.8.5.6 Truth tables

4.12.8.5.6.1 FrameSizeFits rules truth table

Table 379 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 379.

Table 379 – 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	—	—	—	—	—	—

4.12.8.5.6.2 StateChecker truth table

Table 380 contains the truth table used by the MUX.

Table 380 – 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

4.12.8.6 Network Access Receiver Protocol Machine

4.12.8.6.1 Primitive definitions

4.12.8.6.1.1 Primitives exchanged between remote machines

The service primitives including their associated parameters issued or received by Network Access Sender (DEMUX) are described in the service definition and shown in Table 381.

Table 381 – 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	—

4.12.8.6.1.2 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 382.

Table 382 – 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	—	—

4.12.8.6.2 State transition diagram

The state transition diagram of the DEMUX is shown in Figure 107.

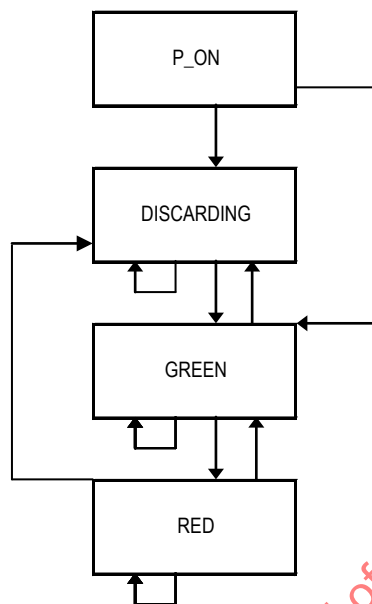


Figure 107 – State transition diagram of DEMUX

States of the DEMUX are:

P_ON	Power on
DISCARDING	port state discarding
GREEN	green period
RED	red period

4.12.8.6.3 State machine description

The DEMUX is responsible for mapping MAC services according to IEEE 802.3 to LMPM, RED_RELAY and MAC_RELAY (IEEE 802.1Q).

The generation of the flag ReceivedInRED shall be done by checking the receive TimeStamp made by the SHIM. If the value of the ReceiveTimeStamp is equal or greater the RedOrangePeriodBegin and less or equal the GreenPeriodBegin point in time, then the ReceivedInRED shall be set to TRUE, otherwise to FALSE.

4.12.8.6.4 DEMUX state table

The DEMUX state table is shown in Table 383.

Table 383 – 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 // 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
14	GREEN	M_UNITDATA.ind () /CTRL => Get_Receive_TimeStamp () DMUX_P_Data.ind ()	GREEN

#	Current State	Event /Condition =>Action	Next State
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

4.12.8.6.5 Functions, Macros, Timers and Variables

Table 384 contains the functions, macros, timers and variables used by the DEMUX and their arguments and their descriptions.

Table 384 – Functions, Macros, Timers and Variables used by the DEMUX

Name	Type	Function/Meaning
Get_Receive_TimeStamp	Function	Get the receive time stamp frame the Timestamp Shim for this frame
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...0xFFFF
FRAGMENT	Macro	Frame using the fragmentation protocol. EtherType == 0x8892 FrameID == Fragmentation
FrameIngressFilter	Macro	Checks whether this frame is filtered by an ingress rule.

Name	Type	Function/Meaning
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) are defined for this specification. It includes 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 385 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 385 shall represent the content of the DLSDU in Table 10.

Table 385 – 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 386 defines structures for substitutions of elements of the APDU structures shown in Table 385.

Table 386 – 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 387.

Table 387 – 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	NDREMapLookupReq or NDREMapLookupFreeReq
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 388.

Table 388 – 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	NDREMapLookupReq or NDREMapLookupFreeReq, 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 389 and Table 390.

Table 389 – 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 390 – 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

Multicast IP address	Associated Multicast MAC address	Associated FrameID	Usage
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 391.

Table 391 – 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 392.

Table 392 – 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. The IO controller uses additional “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 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 393.

Table 393 – 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 394.

Table 394 – 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 108.

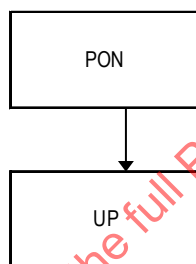


Figure 108 – State transition diagram of ACCM

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 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 into 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 395 contains the state table used by ACCM.

Table 395 – 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 396 contains the functions, macros, timers and variables used by the ACCM and their arguments and their descriptions.

Table 396 – 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 initialize the ARP cache
RemoveStaticARPcacheEntry	Function	This local function removes a static entry to 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 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 397.

High level interfaces may offer a GetHostByName service instead of the QNAME service.

Table 397 – 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 398.

Table 398 – 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 399 contains the functions, macros, timers and variables used by the DNS and their arguments and their descriptions.

Table 399 – 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 ID.

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 400.

Table 400 – Remote primitives issued or received by DHCP

Primitive name	Source	Destination	Associated parameters	Functions
DHCP offer	DHCP	DHCP server	OP, HTYPE, HLEN, HOPS, XID, SECS, FLAGS, CIADDR, YIADDR, SIADDR, GIADDR, CHADDR, DHCPOptions (see IETF RFC 2131 and SuboptionDHCP) ^a	Second service in a sequence to offer the DHCP client a parameter set.
DHCP acknowledgement	DHCP	DHCP server	OP, HTYPE, HLEN, HOPS, XID, SECS, FLAGS, CIADDR, YIADDR, SIADDR, GIADDR, CHADDR, DHCPOptions (see IETF RFC 2131 and SuboptionDHCP) ^a	Fourth service in a sequence to deliver the agreed parameter set.
DHCP discovery	DHCP client	DHCP	OP, HTYPE, HLEN, HOPS, XID, SECS, FLAGS, CIADDR, YIADDR, SIADDR, GIADDR, CHADDR, DHCPOptions (see IETF RFC 2131 and SuboptionDHCP) ^a	First service in a sequence to find a DHCP server.
DHCP request	DHCP client	DHCP	OP, HTYPE, HLEN, HOPS, XID, SECS, FLAGS, CIADDR, YIADDR, SIADDR, GIADDR, CHADDR, DHCPOptions (see IETF RFC 2131 and SuboptionDHCP) ^a	Third service in a sequence which is used by the client to select a server.
^a A manufacturer may add additional DHCPOptions.				

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 401.

Table 401 – Local primitives issued or received by machines

Primitive name	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.15.3 DHCP state transition diagram

See IETF RFC 2131 (DHCP).

4.15.4 State machine description

See IETF RFC 2131 (DHCP).

4.15.5 DHCP state table

See IETF RFC 2131 (DHCP).

4.15.6 Functions, Macros, Timers and Variables

Table 402 contains the functions, macros, timers and variables used by the DHCP and their arguments and their descriptions.

Table 402 – Functions, Macros, Timers and Variables used by the DHCP

Name	Type	Function/Meaning
—	—	—

4.16 Simple network management

4.16.1 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 802.1AB (LLDP-MIB)

are defined for this specification.

Furthermore, the PNIO MIB is specified in 4.16.3 and LLDP EXT MIB is specified in 4.16.6.

SNMP write requests (SNMP_SetRequest) to MIB objects should be rejected, except for the writable OIDs shown in Table 403.

4.16.2 IETF RFC 1213-MIB

OIDs according Table 403 shall be supported if SNMP is supported. OIDs with Accessibility:=Write shall be stored permanently.

Table 403 – List of supported IETF RFC 1213-MIB objects

OID value	OID name	Accessible
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
1.3.6.1.2.1.1.5	sysName	Read / Write
1.3.6.1.2.1.1.6	sysLocation	Read / Write
1.3.6.1.2.1.1.7	sysServices	Read
^a Coding according “SystemIdentification” (preferred) or “ExtendedSystemIdentification” recommended.		

4.16.3 Enterprise number for PNIO MIB

The coding of the field enterprise number shall be according to Table 404.

Table 404 – Enterprise number

Value (decimal)	Meaning
24 686	Enterprise number for the PNIO MIB

4.16.4 MIB cross reference

It is recommended that each (LLDP-MIB.)“lldpLocPortDesc” is defined disjunct to make a cross reference to the IETF RFC 1213-MIB possible.

The link between the IETF RFC 1213-MIB and the LLDP-MIB shall be done according Table 405.

Table 405 – 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 Table 406.

Table 406 – Cross reference – PDPortDataAdjust

PDPortDataAdjust	Rule	IETF RFC 1213-MIB
LinkState.Link == DOWN	==	mib-2.interfaces.ifTable.ifIndex.ifAdminStatus

4.16.5 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.

4.16.6 LLDP EXT MIB

See Annex V

4.17 Common DLL Mapping Protocol Machines**4.17.1 Overview**

The DLL Mapping Protocol Machines (DMPM) consists of several protocol machines:

- Handling of the services invoked by other Application Layer entities (LMPM)
- Forwarding for RT_CLASS_3 (RED_Relay)
- Forwarding for DFP (DFP_Relay)

NOTE 1 Forwarding of standard frames is according to IEEE 802.1Q and therefore not specified here.

- Mapping to Media Access Control interface (MAC) and send/receive functions for synchronization messages

NOTE 2 Invoking of MAC is according to IEEE 802.3 and therefore not specified here.

- Fragmentation and reassembly (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 109 illustrates the structuring of the protocol machines for the DMPM for a device and Annex H describes the upper protocol layers.

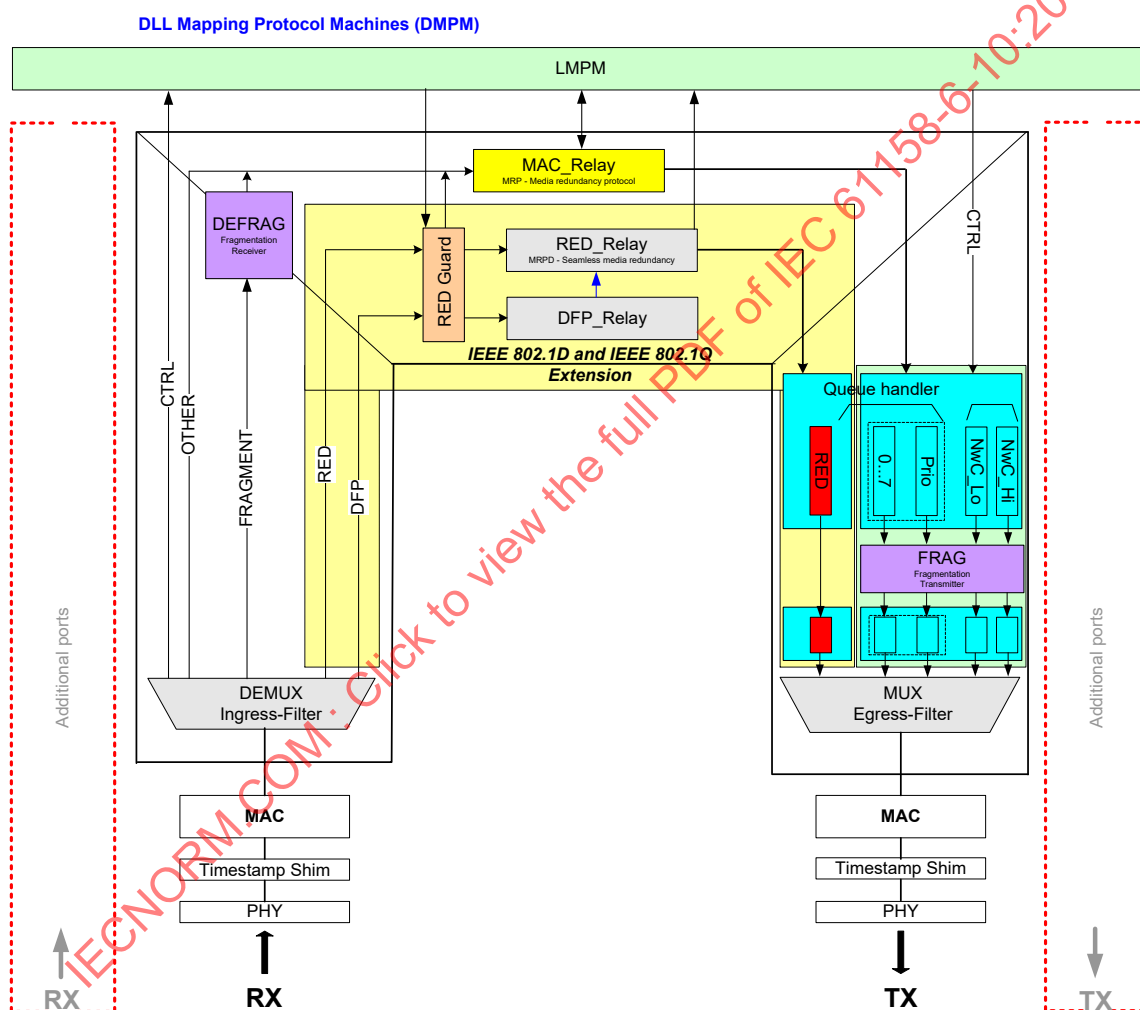


Figure 109 – Structuring of the protocol machines within the DMPM (bridge)

NOTE 3 This picture is often called baggy pants model.

4.17.2 Data Link Layer Mapping Protocol Machine

4.17.2.1 Primitive definitions

4.17.2.1.1 Primitives exchanged between remote machines

The 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 407.

Table 407 – 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 direct 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.
LMPM_P_Data.cnf	LMPM	PTCP	CREP, D_Port, Tstamp,	—

Primitive	Source	Destination	Associated parameters	Functions
			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 IEEE 802.1Q. It searches the filtering database and executes for each found destination port the M_UNITDATA.req.
MUX_Data.ind	MUX	LMPM	CREP, D_Port, Tstamp, Status	Due to the definition of a M_UNITDATA.req a indication is needed the 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.

4.17.2.1.2 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 408.

Table 408 – Local primitives issued or received by LMPM

Primitive	Source	Destination	Associated parameters	Functions
—	—	—	—	—

4.17.2.2 State transition diagram

The state transition diagram of the LMPM is shown in Figure 110.

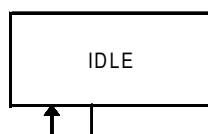


Figure 110 – State transition diagram of LMPM

State of the LMPM is:

IDLE

The LMPM is the interface between the bridges or the MAC and the local state machines.

4.17.2.3 State machine description

The LMPM forms 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.

4.17.2.4 LMPM state table

The LMPM state table is shown in Table 409.

Table 409 – 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	MUX_Data.ind () => ignore LMPM_P_Data.cnf ()	IDLE
3	IDLE	LMPM_N_Data.req () /CheckPDUType () == other //NOTE Other includes RT_CLASS_UDP and RTA_CLASS_UDP => LMPM_N_Data.cnf () MAC_RELAY_Data.req ()	IDLE
4	IDLE	MAC_RELAY_Data.cnf () => ignore	IDLE
5	IDLE	LMPM_A_Data.req () /CheckPDUType () == RTA_CLASS_1 => LMPM_A_Data.cnf () MAC_RELAY_Data.req ()	IDLE

#	Current State	Event /Condition =>Action	Next State
6	IDLE	LMPM_C_Data.req () /CheckPDUType () == RT_CLASS_1 => LMPM_C_Data.cnf () MAC_RELAY_Data.req ()	IDLE
7	IDLE	LMPM_C_Data.req () /CheckPDUType () == RT_CLASS_2 => LMPM_C_Data.cnf () MAC_RELAY_Data.req ()	IDLE
8	IDLE	LMPM_C_Data.req () /CheckPDUType () == RT_CLASS_3 => LMPM_C_Data.cnf () QueueHandler.req ()	IDLE
9	IDLE	DMUX_P_Data.ind () /CheckPDUType () == TimeStamped => LMPM_P_Data.ind ()	IDLE
10	IDLE	RED_RELAY_Data.ind () /CheckPDUType () == RT_CLASS_3 => LMPM_C_Data.ind ()	IDLE
11	IDLE	MAC_RELAY_Data.ind () /CheckPDUType () == RT_CLASS_2 => LMPM_C_Data.ind ()	IDLE
12	IDLE	MAC_RELAY_Data.ind () /CheckPDUType () == RT_CLASS_1 => LMPM_C_Data.ind ()	IDLE
13	IDLE	MAC_RELAY_Data.ind () /CheckPDUType () == RTA_CLASS_1 => LMPM_A_Data.ind ()	IDLE
14	IDLE	MAC_RELAY_Data.ind () /CheckPDUType () == other => LMPM_N_Data.ind ()	IDLE

4.17.2.5 Functions, Macros, Timers and Variables

Table 410 contains the functions, macros, timers and variables used by the LMPM and their arguments and their descriptions.

Table 410 – Functions, Macros, Timers and Variables used by the LMPM

Name	Type	Function/Meaning
CheckPDUType	Function	This local function check the PDU type and returns the values – RT_CLASS_1 – RT_CLASS_2 – RT_CLASS_3 – RTA_CLASS_1 – Other – TimeStamped

4.18 Additional definitions

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, and Annex X give additional information.

5 Application layer protocol specification for decentralized periphery

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 411 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 application

- shall support the highest stated BlockVersionHigh and/or BlockVersionLow shown in Table 411 if not stated otherwise with a GSDML attribute defined for this substitution.
- may support all BlockVersionHigh and/or BlockVersionLow shown in Table 411 or in earlier versions of this specification.

An IO controller application

- shall support the highest stated BlockVersionHigh and/or BlockVersionLow shown in Table 411.
- may support all BlockVersionHigh and/or BlockVersionLow shown in Table 411 or in earlier versions of this specification.

Table 411 – 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 424): BlockHeader, AlarmType, API(=0), SlotNumber(=0x8000), SubslotNumber(=0), ModuleIdentNumber(=0), SubmoduleIdentNumber(=0), AlarmSpecifier, AlarmPayload(=RS_AlarmItem) ^a These fields addresses the submodule which may 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 412.
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 412. 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 416
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*
PROFINETIO-ServiceReqPDU	IODConnectReq ^ IODWriteReq ^ IODWriteMultipleReq ^ IODReadReq ^ IODControlReq ^ IODReleaseReq ^ IOXControlReq
PROFINETIO-ServiceResPDU	IODConnectRes ^ IODWriteRes ^ IODWriteMultipleRes ^ IODReadRes ^ IODControlRes ^ IODReleaseRes ^ IOXControlRes
IODConnectReq with StartupMode:=1	ARBlockReq, {[IOCRBlockReq*], [AlarmCRBlockReq], [ExpectedSubmoduleBlockReq*] ^b , [PrmServerBlock], [MCRBlockReq*] ^a , [ARRPCBlockReq], [IRInfoBlock], [SRInfoBlock], [ARVendorBlockReq*], [ARFSUBlock], [RSInfoBlock] } Special case: "IOCARSingle" or "IOSAR": ARBlockReq, {[IOCRBlockReq*, AlarmCRBlockReq, ExpectedSubmoduleBlockReq*, [PrmServerBlock], [MCRBlockReq*] ^a , [ARRPCBlockReq], [IRInfoBlock] ^d , [ARVendorBlockReq*], [ARFSUBlock], [RSInfoBlock] } Special case: "IOSAR with ARProperties.DeviceAccess:=1": ARBlockReq ^c Special case: "IOCARSR": ARBlockReq, {[IOCRBlockReq*, AlarmCRBlockReq, ExpectedSubmoduleBlockReq*, [PrmServerBlock], [MCRBlockReq*] ^a , [ARRPCBlockReq], SRInfoBlock, [ARVendorBlockReq*], [RSInfoBlock] } ^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.
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 ^a The field ARRPCBlockRes shall only be present if IODConnectReq contains an ARRPCBlockReq. ^b The number of ARVendorBlockRes entries shall be equal to the number of ARVendorBlockReq entries in the IODConnectReq.

Substitution name	Structure
IODConnectReq with StartupMode:=0	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:=0	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
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 PNIOSStatus shall be (0,0,0,0) or a duplicate of the PNIOSStatus 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 PNIOSStatus shall be (0,0,0,0) or a duplicate of the PNIOSStatus from the NDRDataResponse. ^c The value of the parameter PNIOSStatus contains the result of the particular write.
IODWriteResHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding*^a, Index, RecordDataLength, AdditionalValue1, AdditionalValue2, PNIOSStatus, 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

Substitution name	Structure
IODReadReqHeader	BlockHeader, SeqNumber, ARUUUID, API, SlotNumber, SubslotNumber, Padding* ^a , Index, RecordDataLength, TargetARUUUID ^b ^ RWPadding* ^c , RWPadding* ^d ^a The number of padding octets (value=0) in the read request is 2. ^b The optional field TargetARUUUID shall only be used in conjunction with the value 0 of ARUUUID (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.
RecordDataReadQuery	UploadBLOBQuery ^ (BlockHeader, Data* ^a) ^ NULL ^a The usage, structure and values shall be defined for a) Manufacturer specific index range by the manufacturer b) Profile specific index ranges by the profiles c) All other index ranges by this document
IODReadRes	IODReadResHeader, RecordDataRead
IODReadResHeader	BlockHeader, SeqNumber, ARUUUID, 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.

Substitution name	Structure
IOXControlReq with StartupMode:=0	(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.
IOXControlRes with StartupMode:=0	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 [Padding*] behind to ensure Unsigned32 alignment. The BlockType field within the BlockHeader shall be set to ARBlockReq according to Table 412.
ARBlockRes	BlockHeader, ARType, ARUUID, SessionKey, CMResponderMacAdd, ResponderUDPRTPort ^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 412.
IOCRBlockReq	BlockHeader, IOCRType, IOCRReference, LT, IOCRProperties, DataLength, FrameID, SendClockFactor, ReductionRatio, Phase, Sequence, FrameSendOffset^a, DataHoldFactor^b, DataHoldFactor, IOCRTagHeader, IOCRMulticastMACAdd, 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. The BlockType field within the BlockHeader shall be set to IOCRBlockReq according to Table 412. 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.

Substitution name	Structure
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 412.
IOCRBlockRes	BlockHeader, IOCRTYPE, IOCRReference, FrameID The BlockType field within the BlockHeader shall be set to IOCRBlockRes according to Table 412.
ExpectedSubmodule-BlockReq	BlockHeader, NumberOfAPIs, (API, SlotNumber^a, ModuleIdentNumber, ModuleProperties, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber, SubmoduleProperties^b, (DataDescription, SubmoduleDataLength, LengthIOPS, LengthIOCS)*))*^c ^a This parameter is constant for one ExpectedSubmoduleBlockReq because each contains the data for only one slot. ^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 412.
ModuleDiffBlock	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber^a, ModuleState^b, NumberOfSubmodules, [(SubslotNumber, SubmoduleIdentNumber^c, SubmoduleState)*]))*^c ^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 412.
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 412.
AlarmCRBlockRes	BlockHeader, AlarmCRTYPE, LocalAlarmReference, MaxAlarmDataLength The BlockType field within the BlockHeader shall be set to AlarmCRBlockRes according to Table 412.
PrmServerBlock	BlockHeader, ParameterServerObjectUUID, ParameterServerProperties, CMInitiatorActivityTimeoutFactor, StationNameLength, ParameterServerStationName^a ^a An IO device should support a [Padding*] behind to ensure Unsigned32 alignment. The BlockType field within the BlockHeader shall be set to PrmServerBlock according to Table 412.
ARRPCBlockReq	BlockHeader, InitiatorRPCServerPort The BlockType field within the BlockHeader shall be set to ARRPCBlockReq according to Table 412.
ARRPCBlockRes	BlockHeader, ResponderRPCServerPort The BlockType field within the BlockHeader shall be set to ARRPCBlockRes according to Table 412.
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 412.

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 412.
SRInfoBlock	BlockHeader, RedundancyDataHoldFactor, SRProperties The BlockType field within the BlockHeader shall be set to SRInfoBlock according to Table 412.
ARFSUBlock	ARFSUDDataAdjust The BlockType field within the BlockHeader shall be set to ARFSUBlock according to Table 412.
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 412.
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 412. Special case "Extended identification rules" BlockHeader, APStructureIdentifier (:=0x8000), API (:=0), ExpectedExtendedIdentification-Info, [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 412. Special case "Extended identification rules" BlockHeader, APStructureIdentifier (:=0x8000), API (:=0), RealExtendedIdentificationInfo, [Padding*^a]
ExpectedExtended-IdentificationInfo	ExtendedIdentificationInfo
RealExtended-IdentificationInfo	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 412.
PDIRSubframeData	BlockHeader, NumberOfSubframeBlocks, SubframeBlock* The BlockType field within the BlockHeader shall be set to PDIRSubframeBlock according to Table 412.
ControlBlockConnect	BlockHeader, Padding, Padding, ARUUID, SessionKey, Padding, Padding, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to IOBlockReq, IOBlockRes, IOXBlockReq and IOXBlockRes according to Table 412.

Substitution name	Structure
ControlBlockPlug	BlockHeader, Padding, Padding, ARUID, SessionKey, AlarmSequenceNumber, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to IOXBlockReq, IOXBlockRes according to Table 412. The field AlarmSequenceNumber shall contain the value of the sub-field AlarmSpecifier.SequenceNumber of the corresponding AlarmNotification-PDU.
ControlBlockRFC	BlockHeader, Padding, Padding, ARUID, SessionKey, Padding, Padding, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to ReadyForCompanionBlock according to Table 412.
ControlBlockRTC	BlockHeader, Padding, Padding, ARUID, SessionKey, Padding, Padding, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to ReadyForRTCLASS3Block according to Table 412.
ReleaseBlock	BlockHeader, Padding, Padding, ARUID, SessionKey, Padding, Padding, ControlCommand, ControlBlockProperties The BlockType field within the BlockHeader shall be set to ReleaseBlock according to Table 412.
SubmoduleListBlock	BlockHeader, NumberOfEntries, (API, SlotNumber, SubslotNumber)* The BlockType field within the BlockHeader shall be set to SubmoduleListBlock according to Table 412.
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 ^ PDPortFODDataReal ^ PDPortFODDataAdjust ^ PDPortFODDataCheck ^ PDRealData ^ PDExpectedData* ^ PDNCDataCheck ^ PDPortStatistic ^ I&M0FilterData ^ ARFSUDDataAdjust ^ PDInterfaceFSUDDataAdjust ^ PDInterfaceDataReal ^ AutoConfiguration ^ PDIRSubframeData ^ UploadBLOB ^ PE_EntityFilterData ^ PE_EntityStatusData ^ AssetManagementData ^ RS_AdjustObserver ^ RS_GetEvent ^ PDPortMrplcDataAdjust ^ PDPortMrplcDataCheck ^ PDPortMrplcDataReal ^ 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
RecordDataWrite	SubstituteValue ^ IMDataWrite ^ PDPortDataAdjust ^ PDPortDataCheck ^ PDIRData ^ PDSyncData ^ PDTimeData ^ IsochronousModeData ^ PDInterfaceMrpDataAdjust ^ PDInterfaceMrpDataCheck ^ PDPortMrpDataAdjust ^ PDPortFODDataAdjust ^ PDPortFODDataCheck ^ PDNCDataCheck ^ ARFSUDDataAdjust ^ PDInterfaceFSUDDataAdjust ^ PDIRSubframeData ^ PDInterfaceAdjust ^ CombinedObjectContainer ^ RS_AdjustObserver ^ RS_AckEvent ^ PDPortMrplcDataAdjust ^ PDPortMrplcDataCheck ^ 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.

Substitution name	Structure
ChannelDiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType
Manufacturer-SpecificDiagnosis	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.
ExtChannel-DiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType, ExtChannelErrorType, ExtChannelAddValue
QualifiedChannel-Diagnosis	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
ExpectedIdentification-Data 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.

Substitution name	Structure
PdevData	BlockHeader, Padding, Padding, [PDIRData], [PDSyncData], [PDIRSubframeData]
PDRealData ^c	MultipleBlockHeader, { [PDPortDataReal]^{b d}, [PDPortDataRealExtended]^{b d}, [PDInterfaceMrpDataReal], [PDPortMrpDataReal], [PDPortFODDataReal]^a, [PDInterfaceDataReal], [PDPortStatistic], [PDPortMrplcDataReal] } ^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) need its own MultipleBlockHeader ^d If MAUTypeExtension is used, both blocks shall be provided
PDExpectedData ^b	MultipleBlockHeader, { [PDPortDataCheck]^a, [PDPortDataAdjust]^a, [PDInterfaceMrpDataAdjust], [PDInterfaceMrpDataCheck], [PDPortMrpDataAdjust], [PDPortFODDataAdjust], [PDPortFODDataCheck], [PDNCDataCheck], [PDInterfaceFSUDDataAdjust], [PDInterfaceAdjust], [PDPortMrplcDataAdjust], [PDPortMrplcDataCheck] } ^a The fields SlotNumber and SubslotNumber shall be ignored ^b Each submodule's data (for example interface or port) need its own MultipleBlockHeader
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 ^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 640 ^d The number of reserved octets shall be 2. ^e The local calculated LineDelay shall be used.
PDPortDataReal-Extended	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 by means of LLDP received values shall be used.
PDInterfaceDataReal	BlockHeader, LengthOwnStationName, OwnStationName, [Padding*]^a, MACAddressValue^b, [Padding*]^a, IPParameterValue, [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
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 May only be used with StartupMode:=0	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 0 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 0 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 *], [ARFSUBBlock], [SRLData] ^b, [RSInfoBlock] } ^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.
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, InstanceHigh, InstanceLow, DeviceIdentNumber
ParameterServer-ObjectUUID	PROFINETIOConstantValue, InstanceHigh, InstanceLow, DeviceIdentNumber

Substitution name	Structure
DeviceID	DeviceIDHigh, DeviceIDLow
VendorID	VendorIDHigh, VendorIDLow
DeviceIdentNumber	DeviceID, VendorID
IMDataWrite	$[I\&M1] \wedge [I\&M2] \wedge [I\&M3] \wedge [I\&M4]^a$ ^a Should only be used together with Functional Safety
IMData	$I\&M0 \wedge [I\&M1] \wedge [I\&M2] \wedge [I\&M3] \wedge [I\&M4]^a \wedge [I\&M5]$ ^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
I&M3	BlockHeader, IM_Descriptor
I&M4	BlockHeader, IM_Signature
I&M5	BlockHeader, NumberOfEntries, (I&M5Data $\wedge$ 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 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&M0FilterData-Submodule	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

Substitution name	Structure
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.
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 contain the receive path and one the transmit path. If the transmit path doesn't exist (outbound frame is fully consumed by this node) the corresponding entry shall be skipped. ^c With MRPD: A consumer shall exist two times (FrameID and FrameID+1) and a provider shall exist two times (FrameID and FrameID+1) in case of IOCR and may exist more than two times 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 do 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.
AdjustMulticast-Boundary	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
AdjustMAUType-Extension	BlockHeader, Padding, Padding, MAUTypeExtension, AdjustProperties
AdjustLinkState	BlockHeader, Padding, Padding, LinkState, AdjustProperties
AdjustPeerToPeer-Boundary	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.
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

Substitution name	Structure
CheckMAUType-Extension	BlockHeader, MAUTypeExtension
CheckLinkState	BlockHeader, LinkState
CheckSyncDifference	BlockHeader, CheckSyncMode
CheckMAUType-Difference	BlockHeader, MAUTypeMode
PDInterfaceAdjust	BlockHeader, Padding, Padding, MultipleInterfaceMode, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDInterface-MrpDataAdjust 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.
PDInterface-MrpDataAdjust with BlockVersionLow = 1	BlockHeader, Padding, MRP_NumberOfEntries, MrpInstanceDataAdjustBlock*
PDInterface-MrpDataReal 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.
PDInterface-MrpDataReal with BlockVersionLow = 2	BlockHeader, Padding, MRP_NumberOfEntries, MrpInstanceDataRealBlock*
PDInterface-MrpDataCheck with BlockVersionLow = 0	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Check
PDInterface-MrpDataCheck 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

Substitution name	Structure
PDPorMrpDataReal with BlockVersionLow = 0	BlockHeader, Padding, Padding, MRP_DomainUUID
PDPorMrpDataReal 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
MrpInstance-DataAdjustBlock	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.
MrpInstance-DataRealBlock	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.
MrpInstance-DataCheckBlock	BlockHeader, Padding, MRP_Instance, MRP_DomainUUID, MRP_Check
PDPorFODDataReal	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.

Substitution name	Structure
FiberOpticPowerBudget Real	FiberOpticPowerBudgetType ^a ^a FiberOpticPowerBudgetType.CheckEnable shall be set to zero. It has no meaning in this usage.
PDPortFODDataAdjust	BlockHeader, Padding, Padding, FiberOpticType, FiberOpticCableType
PDPortFODDataCheck	BlockHeader, Padding, Padding, MaintenanceRequiredPowerBudget, MaintenanceDemandedPowerBudget, ErrorPowerBudget
MaintenanceRequired-PowerBudget	FiberOpticPowerBudgetType
MaintenanceDemanded-PowerBudget	FiberOpticPowerBudgetType
ErrorPowerBudget	FiberOpticPowerBudgetType
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.
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.
PDInterface-FSUDDataAdjust	BlockHeader, Padding, Padding, { [FSHelloBlock], [FastStartUpBlock] } ^a ^a At least one optional block shall be existing.
ARFSUDDataAdjust	BlockHeader, Padding, Padding, { [FSPParameterBlock], [FastStartUpBlock] } ^a ^a At least one optional block shall be existing.
FSHelloBlock	BlockHeader, Padding, Padding, FSHelloMode, FSHelloInterval, FSHelloRetry, FSHelloDelay
FSPParameterBlock	BlockHeader, Padding, Padding, FSPParameterMode, FSPParameterUUID
FastStartUpBlock	BlockHeader, Padding, Padding, Data*, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
AutoConfiguration	BlockHeader, Padding, Padding, ACCommunication, ACConfiguration, [ACIsochronous], [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
ACCommunication	BlockHeader, Padding, Padding, ACMinDeviceInterval, ACCommunicationProperties, SendClockFactorArray, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
SubmoduleInputData-Length	SubmoduleDataLength
SubmoduleOutputData-Length	SubmoduleDataLength
ACConfiguration	BlockHeader, Padding, Padding, (API, SlotNumber, SubslotNumber, ModuleIdentNumber, SubmoduleIdentNumber, SubmoduleInputDataLength, SubmoduleOutputDataLength)*
ACIsochronous	BlockHeader, Padding, Padding, (API, SlotNumber, SubslotNumber, TimeIOBase ^a , TimeIOInput ^b , TimeIOOutput ^c)* ^a Minimum value according to the GSDML attribute T_IO_Base ^b Minimum value according to the GSDML attribute T_IO_InputMin ^c Minimum value according to the GSDML attribute T_IO_OutputMin

Substitution name	Structure
CombinedObject-Container	(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, [Padding]* ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
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_HardwareOnly-Information	IM_UniqueIdentifier, AM_Location, IM_Annotation, IM_OrderID, AM_HardwareRevision, IM_Serial_Number, AM_DeviceIdentification, AM_TypeIdentification, IM_Hardware_Revision
AM_FirmwareOnly-Information	IM_UniqueIdentifier, AM_Location, IM_Annotation, IM_OrderID, AM_SoftwareRevision, IM_Serial_Number, IM_Software_Revision, AM_DeviceIdentification, AM_TypeIdentification
RS_AdjustObserver	BlockHeader, RS_AdjustInfo
RS_AdjustInfo	NumberOfEntries, RS_AdjustBlock*
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.

Substitution name	Structure
RS_EventData-Extension	RS_ExtensionBlockType, RS_ExtensionBlockLength, Data*, [Padding]*^a Special case "RSO_BufferObserver": — Special case "RSO_StatusObserver": RS_ExtensionBlockType(=0x80), RS_ExtensionBlockLength, RS_BlockType^b, RS_ReasonCode Special case "RSO_TimeStatus": RS_ExtensionBlockType(=0x80), RS_ExtensionBlockLength, [Padding]*^a, RS_DomainIdentification, RS_MasterIdentification, [Time_TimeStamp]^c 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 of time takes place.
RS_IdentificationInfo	AM_DeviceIdentification, IM_Tag_Function, IM_Tag_Location
RS_AckEvent	BlockHeader, RS_AckInfo
RS_AckInfo	RS_Specifier
PDPorMrplcDataAdjust	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.
PDPorMrplcDataCheck	BlockHeader, Padding, Padding, MRPIC_Check
PDPorMrplcDataReal	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_StateData	MRPIC_State
SystemIdentification	DeviceType, Blank, OrderID, Blank, IM_Serial_Number, Blank, HWRevision, Blank, SWRevisionPrefix, SWRevision
ExtendedSystem-Identification	OctetString[64], SystemIdentification

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 412. 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 the vendor specific portion.

Table 412 – 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

Value (hexadecimal)	Used for	Used by
0x001B	SRLData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. ARDataInfo.SRLData
0x001C – 0x001F	Reserved	—
0x0020	I&M0	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. IMData.I&M0
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 – 15 ^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

Value (hexadecimal)	Used for	Used by
0x0036	Asset Management – Full information	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. AssetManagementData.AM_FullInformation PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. I&M5.AM_FullInformation
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 – 0x010F	Reserved	—
0x0110	IODBlockReq, shall only be used in conjunction with connection establishment phase	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 within 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)
0x0118 – 0x01FF	Reserved	—
0x0200	PDPorDataCheck	PROFINETIO- ServiceReqPDU.IODReadReq.RecordDataRead. PDPorDataCheck PROFINETIO- ServiceReqPDU.IODWriteReq.RecordDataWrite. PDPorDataCheck PROFINETIO- ServiceReqPDU.IODReadReq.RecordDataRead. PDExpectedData.PDPorDataCheck
0x0201	PdevData	PROFINETIO- ServiceReqPDU.IODReadReq.RecordDataRead. PdevData
0x0202	PDPorDataAdjust	PROFINETIO- ServiceReqPDU.IODReadReq.RecordDataRead. PDPorDataAdjust PROFINETIO- ServiceReqPDU.IODWriteReq.RecordDataWrite. PDPorDataAdjust PROFINETIO- ServiceReqPDU.IODReadReq.RecordDataRead. PDExpectedData.PDPorDataAdjust
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
0x020F	PDPortDataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPortDataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDRealData.PDPortDataReal

Value (hexadecimal)	Used for	Used by
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
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

Value (hexadecimal)	Used for	Used by
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	PDPorFODataReal	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
0x0222	PDPorFODataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPortFODataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDPortFODataAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDExpectedData.PDPortFODataAdjust

Value (hexadecimal)	Used for	Used by
0x0223	PDPorFODataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorFODataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDPorFODataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDExpectedData.PDPorFODataCheck
0x0224	AdjustPeerToPeerBoundary Sub block for adjusting the peer to peer boundary	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorDataAdjust.AdjustPeerToPeerBoundary PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDPorDataAdjust.AdjustPeerToPeerBoundary
0x0225	AdjustDCPBoundary Sub block for adjusting the DCP boundary	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorDataAdjust.AdjustDCPBoundary PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDPorDataAdjust.AdjustDCPBoundary
0x0226	AdjustPreambleLength Sub block for adjusting the used length of preamble	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorDataAdjust.AdjustPreambleLength PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDPorDataAdjust.AdjustPreambleLength
0x0227	CheckMAUTypeExtension Sub block for checking MAUTypeExtension	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorDataCheck. CheckMAUTypeExtension PROFINETIO-ServiceReqPDU.IODWriteReq. RecordDataWrite. PDPorDataCheck. CheckMAUTypeExtension
0x0228	FiberOpticDiagnosisInfo Sub block for reading real fiber optic diagnosis data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorFODataReal.FiberOpticDiagnosisInfo PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDRRealData.FiberOpticDiagnosisInfo
0x0229	AdjustMAUTypeExtension Sub block for adjusting MAUTypeExtension	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorDataAdjust. AdjustMAUTypeExtension PROFINETIO-ServiceReqPDU.IODWriteReq. RecordDataWrite. PDPorDataAdjust. AdjustMAUTypeExtension
0x022A	PDIRSubframeData	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDIRSubframeData PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDIRSubframeData PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PdevData.PDIRSubframeData

Value (hexadecimal)	Used for	Used by
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 – 0x022F	Reserved	—
0x0230	PDNCDataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDNCDataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite.PDNCDataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead.PDExpectedData.PDNCDataCheck
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

Value (hexadecimal)	Used for	Used by
0x0234	PDPorMrplcDataAdjust	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorMrplcDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDPorMrplcDataAdjust PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDExpectedData.PDPorMrplcDataAdjust
0x0235	PDPorMrplcDataCheck	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorMrplcDataCheck PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDPorMrplcDataCheck PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDExpectedData.PDPorMrplcDataCheck
0x0236	PDPorMrplcDataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorMrplcDataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDRealData.PDPorMrplcDataReal
0x0237 – 0x23F	Reserved	—
0x0240	PDInterfaceDataReal	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDInterfaceDataReal PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDRealData.PDInterfaceDataReal
0x0241 – 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	PDPorStatistic	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorStatistic PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDRealData.PDPorStatistic
0x0252 – 0x025F	Reserved	—
0x0260	OwnPort Sub block of PDPorDataRealExtended	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPorDataReal.OwnPort PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDRealData.PDPorDataRealExtended.OwnPort

Value (hexadecimal)	Used for	Used by
0x0261	Neighbors Sub block of PDPortDataRealExtended	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDPortDataRealExtended.Neighbors PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDRealData.PDPortDataRealExtended.Neighbors
0x0262 – 0x03FF	Reserved	—
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 – 0x05FF	Reserved	—
0x0600	FSHelloBlock	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDInterfaceFSUDDataAdjust.FSHelloBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDInterfaceFSUDDataAdjust.FSHelloBlock PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDExpectedData.PDInterfaceFSUDDataAdjust.FSHelloBlock
0x0601	FSPParameterBlock	PROFINETIO-ServiceReqPDU.IODConnectReq.IOCRBlockReq. ARFSUBlock.ARFSUDDataAdjust.FSPParameterBlock
0x0602	FastStartUpBlock	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDInterfaceFSUDDataAdjust.FastStartUpBlock PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. PDInterfaceFSUDDataAdjust.FastStartUpBlock PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. PDExpectedData.PDInterfaceFSUDDataAdjust. FastStartUpBlock PROFINETIO-ServiceReqPDU.IODConnectReq.IOCRBlockReq. ARFSUBlock.ARFSUDDataAdjust.FastStartUpBlock PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. ARFSUDDataAdjust PROFINETIO-ServiceReqPDU.IODWriteReq.RecordDataWrite. ARFSUDDataAdjust PROFINETIO-ServiceReqPDU.IODConnectReq.ARFSUBlock. ARFSUDDataAdjust
0x0603 – 0x0607	Reserved for FastStartUp	—

Value (hexadecimal)	Used for	Used by
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	AutoConfiguration Auto configuration data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. AutoConfiguration
0x0701	AutoConfiguration Auto configuration communication data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. AutoConfiguration.ACCommunication
0x0702	AutoConfiguration Auto configuration configuration data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. AutoConfiguration.ACConfiguration
0x0703	AutoConfiguration Auto configuration isochronous data	PROFINETIO-ServiceReqPDU.IODReadReq.RecordDataRead. AutoConfiguration.ACIsynchronous
0x0704 – 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
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

Value (hexadecimal)	Used for	Used by
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	Maintenanceltem	RTA-SDU.Alarmnotification-PDU.AlarmPayload.Maintenanceltem
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 – 0x6FFF	Reserved	—
0x7000 – 0x7FFF	Reserved for vendor specific 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	—
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

Value (hexadecimal)	Used for	Used by
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
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 413.

Table 413 – 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 414.

Table 414 – 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 415.

Table 415 – 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

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019

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 416. AlarmType and UserStructureIdentifier combinations are shown in Table 572.

Table 416 – 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	Alarm_Low	No	Optional	Yes	Normative

Value (hexadecimal)	Meaning	Alarm priority	Attached to the Diagnosis ASE	AlarmPayload required	USI structured AlarmPayload	Allowed USI range
	Logical "Pull" of a submodule to withdraw ownership					
0x0009	Released Logical "Plug" of a submodule to return ownership or trigger a reparameterization	Alarm_Low	No	Optional	Yes	Normative
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

Value (hexadecimal)	Meaning	Alarm priority	Attached to the Diagnosis ASE	AlarmPayload required	USI structured AlarmPayload	Allowed USI range
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	—	—	—	—	—
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 CMC TL.

^d An AlarmPayload may not exist if a diagnosis disappears. 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 417. 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 may start for each AR of an AR set with every switchover with an arbitrary value.

Whenever an Alarm is repeated over different ARs of an ARset the same SequenceNumber shall be used.

Table 417 – AlarmSpecifier.SequenceNumber

Value (hexadecimal)	Meaning
0x0000 – 0x07FF	Allowed value range

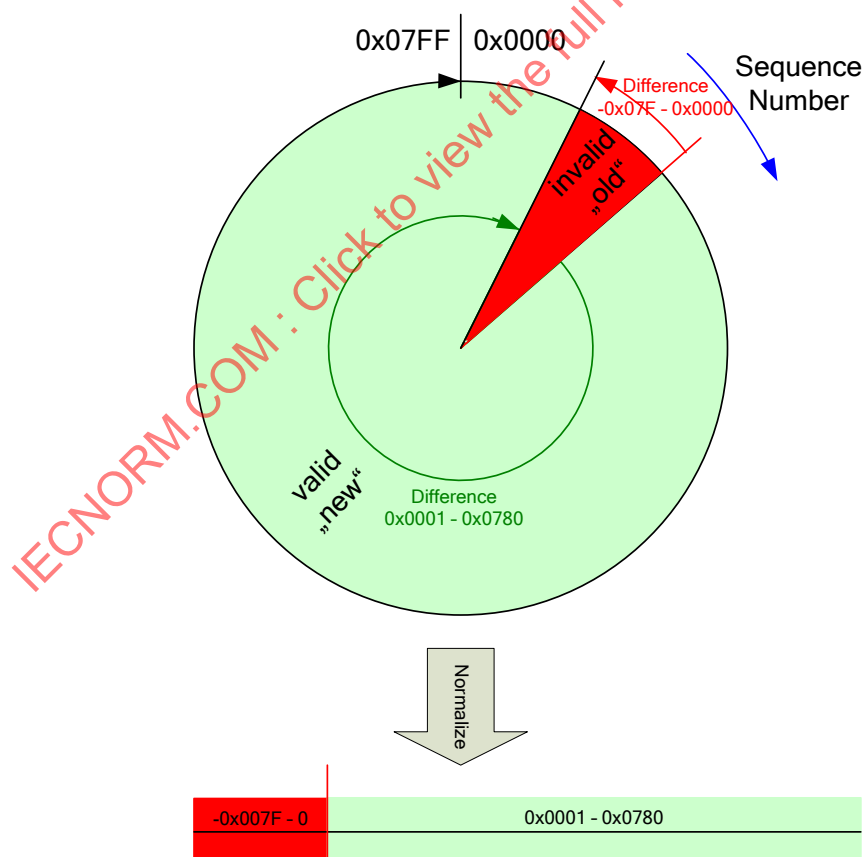


Figure 111 – AlarmSpecifier.SequenceNumber value range

Table 418 defines a maximum SequenceNumber window between an old and a new alarm.

Table 418 – AlarmSpecifier.SequenceNumber Difference

Value (hexadecimal)	Meaning	Use
-0x007F – 0x0000	Red sector according to Figure 111 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 111 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 419. For all other AlarmTypes this field shall be set to zero.

Table 419 – 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 420. For all other AlarmTypes this field shall be set to zero.

Table 420 – 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 421. For all other AlarmTypes this field shall be set to zero.

Table 421 – 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 exist 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 614	

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 422. For all other AlarmTypes this field shall be set to zero.

Table 422 – 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	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 423.

Table 423 – API

Value (hexadecimal)	Meaning of API
0	Default
0x00000001 – 0x0000FFFF	Used for profile definitions fitting for 5.2.7.9
0x00010000 – 0xFFFFFFFF	Used for profile definitions beyond 5.2.7.9

NOTE API means Application Process Identifier.

5.2.3.2 Coding of the field SlotNumber

This field shall be coded as data type Unsigned16 according to Table 424.

Table 424 – 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 425.

Table 425 – 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) Useable 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 with up to 255 ports. There may be gaps. See Formula (49) If used by system redundancy, the interface submodule mounted left or up into the rack.
	Used for 16 interface submodules with up to 255 ports. There may be gaps. See Formula (49) Used by system redundancy with redundant interface for the interface submodule mounted right or down into the rack.
0x9000 – 0x9FFF	
0xA000 – 0xFFFF	Reserved

The subslotnumber shall be evaluated with 0xLIPP with I counting interfaces (PP == 00 means interface itself) and PP counting ports as shown in Formula (49).

$$\text{SubslotNumber} = 0xL000 + 0x0I00 + 0x00PP \quad (49)$$

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 port of the interface starting with one. The value zero means no port.

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 426. 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 426 – 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 – 0xFFFF	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 evaluating of the address parameters according to the index range.

The rules to evaluate the address parameter shown in Table 427, Table 428, Table 429, Table 430, and Table 431 shall be applied.

NOTE Implicit means from outside an existing AR. Explicit means from inside the existing AR, including DeviceAccess.

Table 427 – 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 428 – 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 429 – 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 430 – 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 431 – 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 432. The identifier is used in Table 434, Table 435, Table 436, Table 437 and Table 438.

Table 432 – 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)

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019

5.2.3.4.4 Record index

A NULL response for a read record request shall 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 433 and the usage of the address parameters according to Table 427.

Table 433 – Index (user specific)

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0 – 0x7FFF	User specific RecordData	All	—	User specific definition	User specific definition	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 434 and the usage of the address parameters according to Table 427.

Table 434 – Index (subslot specific)

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0x8000	ExpectedIdentificationData for one subslot	All	Needed	Possible	Read	No	—	—	—
0x8001	RealIdentificationData for one subslot	All	—	—	Read	No	—	—	—
0x8002 – 0x8009	Reserved	—	—	—	—	—	—	—	—
0x800A	DiagnosisData Diagnosis in channel coding for one subslot	All	Filter	Possible	Read	No	—	—	—
0x800B	DiagnosisData Diagnosis in all codings for one subslot	All	Filter	Possible	Read	No	—	—	—

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0x800C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one subslot	All	Filter	Possible	Read	No	—	—	—
0x800D – 0x800F	Reserved	—	—	—	—	—	—	—	—
0x8010	DiagnosisData Maintenance required in channel coding for one subslot	All	Filter	Possible	Read	No	—	—	—
0x8011	DiagnosisData Maintenance demanded in channel coding for one subslot	All	Filter	Possible	Read	No	—	—	—
0x8012	DiagnosisData Maintenance required in all codings for one subslot	All	Filter	Possible	Read	No	—	—	—
0x8013	DiagnosisData Maintenance demanded in all codings for one subslot	All	Filter	Possible	Read	No	—	—	—
0x8014 – 0x801D	Reserved	—	—	—	—	—	—	—	—
0x801E	SubstituteValue for one subslot	All	—	Possible	Read / Write	No	No	—	Yes
0x801F	Reserved	—	—	—	—	—	—	—	—
0x8020	PDIRSubframeData for one subslot	Interface	—	Possible	Read / Write	No	No	Yes	Yes
0x8021 – 0x8026	Reserved	—	—	—	—	—	—	—	—
0x8027	PDPortDataRealExtended for one subslot	Port	—	—	Read	No	—	—	—
0x8028	RecordInputDataObjectElement for one subslot	All	—	Possible	Read	No	—	—	—
0x8029	RecordOutputDataObjectElement for one subslot	All	—	—	Read	No	—	—	—
0x802A	PDPortDataReal for one subslot	Port	—	—	Read	No	—	—	—
0x802B	PDPortDataCheck for one subslot	Port	—	Possible	Read / Write	No	No	Yes	Yes
0x802C	PDIRData for one subslot	Interface	—	Possible	Read / Write	No	No	Yes	Yes
0x802D	PDSyncData for one subslot with SyncID value 0	Interface	—	Possible	Read / Write	No	No	Yes	Yes

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0x802E	Reserved (legacy)	—	—	—	—	—	—	—	—
0x802F	PDPorDataAdjust for one subslot	Port	—	Possible	Read / Write	No	No	Yes	Yes
0x8030	IsosynchronousModeData for one subslot	All	—	Possible	Read / Write	No	No	Yes	Yes
0x8031	PDTTimeData for one subslot	Interface	—	Possible	Read / Write	No	No	Yes	Yes
0x8032 – 0x804F	Reserved	—	—	—	—	—	—	—	—
0x8050	PDInterfaceMrpDataReal for one subslot	Interface	—	Possible	Read	No	—	—	—
0x8051	PDInterfaceMrpDataCheck for one subslot	Interface	—	Possible	Read / Write	No	No	Yes	Yes
0x8052	PDInterfaceMrpDataAdjust for one subslot	Interface	—	Possible	Read / Write	No	No	Yes	Yes
0x8053	PDPorMrpDataAdjust for one subslot	Port	—	Possible	Read / Write	No	No	Yes	Yes
0x8054	PDPorMrpDataReal for one subslot	Port	—	Possible	Read	No	—	—	—
0x8055	PDPorMrpplcDataAdjust for one subslot	Port	—	Possible	Read / Write	No	No	Yes	Yes
0x8056	PDPorMrpplcDataCheck for one subslot	Port	—	Possible	Read / Write	No	No	Yes	Yes
0x8057	PDPorMrpplcDataReal for one subslot	Port	—	Possible	Read	No	—	—	—
0x8058 – 0x805F	Reserved	—	—	—	—	—	—	—	—
0x8060	PDPorFODDataReal for one subslot	Port	—	—	Read	No	—	—	—
0x8061	PDPorFODDataCheck for one subslot	Port	—	Possible	Read / Write	No	No	Yes	Yes
0x8062	PDPorFODDataAdjust for one subslot	Port	—	Possible	Read / Write	No	No	Yes	Yes
0x8063 – 0x806F	Reserved	—	—	—	—	—	—	—	—
0x8070	PDNCDataCheck for one subslot	Interface	—	Possible	Read / Write	No	No	Yes	Yes
0x8071	PDInterfaceAdjust for one subslot	Interface	—	Possible	Read / Write	No	No	Yes	Yes
0x8072	PDPorStatistic for one subslot	Interface, Port	—	—	Read	No	—	—	—
0x8073 – 0x807F	Reserved	—	—	—	—	—	—	—	—
0x8080	PDInterfaceDataReal for one subslot	Interface	—	—	Read	No	—	—	—
0x8081 – 0x808F	Reserved	—	—	—	—	—	—	—	—
0x8090	PDInterfaceFSUDataAdjust	Interface	—	Possible	Read / Write	No	No	Yes	Yes
0x8091 – 0x809F	Reserved	—	—	—	—	—	—	—	—
0x80A0	Profiles covering energy saving – Record_0	All	—	—	Read / Write	No	No	—	Yes

Value (hexadecimal)	Meaning of index	Applicable Submodules	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0x80A1 – 0x80AE	Reserved for profiles covering energy saving	—	—	—	—	—	—	—	—
0x80AF	PE_EntityStatusData for one subslot	All	—	Possible	Read	No	—	—	—
0x80B0	CombinedObjectContainer	All	—	—	Write	No	—	—	—
0x80B1 – 0x80BF	Reserved	—	—	—	—	—	—	—	—
0x80C0 – 0x80CE	Reserved for reporting system	—	—	—	—	—	—	—	—
0x80CF	RS_AdjustObserver	All	—	Possible	Read / Write	No	No	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	—	—	—	—	—	—	—	—
0x80F0 – 0xAFEF	Reserved	—	—	—	—	—	—	—	—
0xAFF0	I&M0	All	—	—	Read	No	—	—	—
0xAFF1	I&M1	All	—	Possible ^a	Read / Write	No	No	Yes	Yes
0xAFF2	I&M2	All	—	Possible ^a	Read / Write	No	No	Yes	—
0xAFF3	I&M3	All	—	Possible ^a	Read / Write	No	No	Yes	Yes
0xAFF4	I&M4	All	—	Possible ^a	Read / Write	No	No	Yes	—
0xAFF5	I&M5	All	—	—	Read	No	—	—	—
0xAFF6 – 0xAFFF	I&M6 – I&M15 ^b	All	—	—	—	—	—	—	—
0xB000 – 0xBFFF	Reserved for profiles	All	—	—	—	—	—	—	—

Needed:

Reference to an AR needed.

Any service without this reference shall return an error.

Filter:

AR context, Expected identification, is used as filter for the response.

Any service without this reference returns data according RealIdentification.

^a See 5.2.7.2

^b Reserved for additional identification and maintenance data

The coding for module specific defined record data shall be according to Table 435 and the usage of the address parameters according to Table 428.

Table 435 – Index (slot specific)

Value (hexadecimal)	Meaning of index	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0xC000	ExpectedIdentificationData for one slot	Needed	Possible	Read	No	—	—	—
0xC001	RealIdentificationData for one slot	Filter ^a	—	Read	No	—	—	—
0xC002 – 0xC009	Reserved	—	—	—	—	—	—	—
0xC00A	DiagnosisData Diagnosis in channel coding for one slot	Filter	Possible	Read	No	—	—	—
0xC00B	DiagnosisData Diagnosis in all codings for one slot	Filter	Possible	Read	No	—	—	—
0xC00C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one slot	Filter	Possible	Read	No	—	—	—
0xC00D – 0xC00F	Reserved	—	—	—	—	—	—	—
0xC010	DiagnosisData Maintenance required in channel coding for one slot	Filter	Possible	Read	No	—	—	—
0xC011	DiagnosisData Maintenance demanded in channel coding for one slot	Filter	Possible	Read	No	—	—	—
0xC012	DiagnosisData Maintenance required in all codings for one slot	Filter	Possible	Read	No	—	—	—
0xC013	DiagnosisData Maintenance demanded in all codings for one slot	Filter	Possible	Read	No	—	—	—
0xC014 – 0xCFFF	Reserved	—	—	—	—	—	—	—
0xD000 – 0xDFFF	Reserved for profiles	—	—	—	—	—	—	—

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 436 and the usage of the address parameters according to Table 429.

Table 436 – Index (AR specific)

Value (hexadecimal)	Meaning of index	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0xE000	ExpectedIdentificationData for one AR	Needed	Possible	Read	No	—	—	—
0xE001	RealIdentificationData for one AR	Needed	Possible	Read	No	—	—	—
0xE002	ModuleDiffBlock for one AR	Needed	Possible	Read	No	—	—	—
0xE003 – 0xE009	Reserved	—	—	—	—	—	—	—
0xE00A	DiagnosisData Diagnosis in channel coding for one AR	Needed	Possible	Read	No	—	—	—
0xE00B	DiagnosisData Diagnosis in all codings for one AR	Needed	Possible	Read	No	—	—	—
0xE00C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one AR	Needed	Possible	Read	No	—	—	—
0xE00D – 0xE00F	Reserved	—	—	—	—	—	—	—
0xE010	DiagnosisData Maintenance required in channel coding for one AR	Needed	Possible	Read	No	—	—	—

Value (hexadecimal)	Meaning of index	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0xE011	DiagnosisData Maintenance demanded in channel coding for one AR	Needed	Possible	Read	No	—	—	—
0xE012	DiagnosisData Maintenance required in all codings for one AR	Needed	Possible	Read	No	—	—	—
0xE013	DiagnosisData Maintenance demanded in all codings for one AR	Needed	Possible	Read	No	—	—	—
0xE014 – 0xE02F	Reserved	—	—	—	—	—	—	—
0xE030	PE_EntityFilterData for one AR	Needed	Possible	Read	No	—	—	—
0xE031	PE_EntityStatusData for one AR	Needed	Possible	Read	No	—	—	—
0xE032 – 0xE03F	Reserved	—	—	—	—	—	—	—
0xE040	WriteMultiple	—	—	Write	—	No	—	—
0xE041 – 0xE04F	Reserved	—	—	—	—	—	—	—
0xE050	ARFSUDDataAdjust data for one AR Legacy, used only for ARProperties.StartupMode == Legacy	Needed	Possible	Read / Write	No	No	—	—
0xE051 – 0xE05F	Reserved for FastStartUp	—	—	—	—	—	—	—
0xE060	RS_GetEvent	Needed	Possible	Read	No	—	—	—
0xE061	RS_AckEvent	Needed	—	Write	—	No	—	—
0xE062 – 0xEBFF	Reserved	—	—	—	—	—	—	—
0xEC00 – 0xEFFF	Reserved for profiles	Needed	—	—	—	—	—	—
Needed: Reference to an AR needed. Any service without this reference shall return an error.								

The coding for API specific defined record data shall be according to Table 437 and the usage of the address parameters according to Table 430.

Table 437 – Index (API specific)

Value (hexadecimal)	Meaning of index	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0xF000	RealIdentificationData for one API	—	—	Read	No	—	—	—
0xF001 – 0xF009	Reserved	—	—	—	—	—	—	—
0xF00A	DiagnosisData Diagnosis in channel coding for one API	—	Possible	Read	No	—	—	—
0xF00B	DiagnosisData Diagnosis in all codings for one API	—	Possible	Read	No	—	—	—
0xF00C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one API	—	Possible	Read	No	—	—	—
0xF00D – 0xF00F	Reserved	—	—	—	—	—	—	—
0xF010	DiagnosisData Maintenance required in channel coding for one API	—	Possible	Read	No	—	—	—
0xF011	DiagnosisData Maintenance demanded in channel coding for one API	—	Possible	Read	No	—	—	—
0xF012	DiagnosisData Maintenance required in all codings for one API	—	Possible	Read	No	—	—	—
0xF013	DiagnosisData Maintenance demanded in all codings for one API	—	Possible	Read	No	—	—	—
0xF014 – 0xF01F	Reserved	—	—	—	—	—	—	—
0xF020	ARData for one API	—	Possible	Read	No	—	—	—
0xF021 – 0xF03F	Reserved	—	—	—	—	—	—	—
0xF400 – 0xF7FF	Reserved for profiles	—	—	—	—	—	—	—

The coding for device specific defined record data shall be according to Table 438 and the usage of the address parameters according to Table 431.

Table 438 – Index (device specific)

Value (hexadecimal)	Meaning of index	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0xF800 – 0xF80B	Reserved	—	—	—	—	—	—	—
0xF80C	DiagnosisData Diagnosis, Maintenance, Qualified and Status for one device	—	Possible	Read	No	—	—	—
0xF80D – 0xF81F	Reserved	—	—	—	—	—	—	—
0xF820	ARData	—	Possible	Read	No	—	—	—
0xF821	APIData	—	—	Read	No	—	—	—
0xF822 – 0xF82F	Reserved	—	—	—	—	—	—	—
0xF830	LogBookData	—	Possible	Read	No	—	—	—
0xF831	PdevData	—	Possible	Read	No	—	—	—
0xF832 – 0xF83F	Reserved	—	—	—	—	—	—	—
0xF840	I&M0FilterData	—	—	Read	No	—	—	—
0xF841	PDRealData	—	—	Read	No	—	—	—
0xF842	PDExpectedData	—	Possible	Read	No	—	—	—
0xF843 – 0xF84F	Reserved	—	—	—	—	—	—	—
0xF850	AutoConfiguration	—	—	Read	No	—	—	—
0xF851 – F85F	Reserved	—	—	—	—	—	—	—
0xF860	GSD upload using UploadBLOBQuery and UploadBLOB	—	—	Read	No	—	—	—
0xF861 – 0xF86F	Reserved for controller to controller communication	—	—	—	—	—	—	—
0xF870	PE_EntityFilterData	—	Possible	Read	No	—	—	—
0xF871	PE_EntityStatusData	—	Possible	Read	No	—	—	—
0xF872 – 0xF87F	Reserved	—	—	—	—	—	—	—

Value (hexadecimal)	Meaning of index	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0xF880	AssetManagementData Contains all (if only this record is supported) or the first chunk of complete assets (if 0xF881 is supported)	—	Possible	Read	No	—	—	—
0xF881	AssetManagementData Contains the second chunk of complete assets	—	Possible	Read	No	—	—	—
0xF882	AssetManagementData Contains the third chunk of complete assets (if 0xF881 is supported)	—	Possible	Read	No	—	—	—
0xF883	AssetManagementData Contains the fourth chunk of complete assets (if 0xF882 is supported)	—	Possible	Read	No	—	—	—
0xF884	AssetManagementData Contains the fifth chunk of complete assets (if 0xF883 is supported)	—	Possible	Read	No	—	—	—
0xF885	AssetManagementData Contains the sixth chunk of complete assets (if 0xF884 is supported)	—	Possible	Read	No	—	—	—
0xF886	AssetManagementData Contains the seventh chunk of complete assets (if 0xF885 is supported)	—	Possible	Read	No	—	—	—
0xF887	AssetManagementData Contains the eighth chunk of complete assets (if 0xF885 is supported)	—	Possible	Read	No	—	—	—
0xF888	AssetManagementData Contains the ninth chunk of complete assets (if 0xF885 is supported)	—	Possible	Read	No	—	—	—
0xF889	AssetManagementData Contains the tenth chunk of complete assets (if 0xF885 is supported)	—	Possible	Read	No	—	—	—
0xF88A – 0xFBFE	Reserved	—	—	—	—	—	—	—

Value (hexadecimal)	Meaning of index	ARUID context	NULL response	Access	Partial Read	Partial Write	Persistence	Startup parameter
0xFBFF	Trigger index for the RPC connection monitoring inside the CMSM	—	Possible	Read	No	—	—	—
0xFC00 – 0xFFFF	Reserved for profiles	—	—	—	—	—	—	—

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019

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 439. The field RecordDataLength shall only contain the number of user octets.

Table 439 – 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 allows 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 440.

Table 440 – ARTYPE

Value (hexadecimal)	Meaning	Use
0x0000	Reserved	—
0x0001	IOCARSingl	—
0x0002 – 0x0005	Reserved	—
0x0006	IOSAR	The supervisor AR is a special form of the IOCARSingl allowing takeover of the ownership of a submodule
0x0007 – 0x000F	Reserved	—
0x0010	IOCARSingl using RT_CLASS_3	This is a special form of the IOCARSingl indicating RT_CLASS_3 communication
0x0011 – 0x001F	Reserved	—
0x0020	IOCARSR	The SR AR is a special form of the IOCARSingl 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 802.

5.2.4.6 Coding of the field IOCRMulticastMACAdd

This field shall be coded as data type OctetString[6]. The value of the field IOCRMulticastMACAdd shall be set according to the 48-bit universal LAN MAC addresses of IEEE 802, to Table 441 and Table 442.

The IP Multicast address used for multicast communication relation using RT_CLASS_UDP shall be set according IETF RFC 2365.

Table 441 – 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 442 – 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
03-00-00 to F3-FF-00	00-00-00	Fast Forwarding local administered multicast address range

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
Other locally administered multicast	—	Reserved for further multicast addresses within the Type 10 context

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

The Organizationally Unique Identifier (OUI) as defined by IEEE 802 for Type 10 is 00-0E-CF and shall be set according to Table 443.

Table 443 – 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 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 444.

Table 444 – ARProperties.State

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Active
0x02 – 0x07	Reserved

Bit 3: ARProperties.SupervisorTakeoverAllowed

This field shall be set according to Table 445.

Table 445 – ARProperties.SupervisorTakeoverAllowed

Value (hexadecimal)	Meaning
0x00	Not allowed
0x01	Allowed

Bit 4: ARProperties.ParameterizationServer

This field shall be set according to Table 446.

Table 446 – ARProperties.ParameterizationServer

Value (hexadecimal)	Meaning
0x00	External PrmServer
0x01	CM Initiator

Bit 5 – 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 447.

Table 447 – ARProperties.DeviceAccess

Value (hexadecimal)	Meaning
0x00	Only the submodules from the ExpectedSubmoduleBlock are accessible
0x01	Submodule access is controlled by IO device application

Bit 9-10: ARProperties.CompanionAR

This field shall be set according to Table 448.

Table 448 – ARProperties.CompanionAR

Value (hexadecimal)	Meaning	Use
0x00	Single AR	RT_CLASS_1, RT_CLASS_2, RT_CLASS_3, or RT_CLASS_UDP connection establishment
0x01	First AR of a companion pair and a companion AR shall follow	For future use
0x02	Companion AR	For future use
0x03	Reserved	—

Bit 11: ARProperties.AcknowledgeCompanionAR

This field shall be set according to Table 449.

Table 449 – 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, or RT_CLASS_UDP connection establishment
0x01	Companion AR with acknowledge	For future use

Bit 12 – 23: ARProperties.Reserved_2

This field shall be set according to 3.4.2.2.

Bit 24 – 28: ARProperties.Reserved_3

This field shall be set to zero.

Bit 29: ARProperties.CombinedObjectContainer

This field shall be set according to Table 450 and Table 451.

**Table 450 – ARProperties.CombinedObjectContainer with
ARProperties.StartupMode == Legacy**

Value (hexadecimal)	Meaning	Use
0x00	NoCOC	CombinedObjectContainer not used
0x01	—	Reserved

**Table 451 – ARProperties.CombinedObjectContainer with
ARProperties.StartupMode == Advanced**

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 452.

Table 452 – ARProperties.StartupMode

Value (hexadecimal)	Meaning	Use
0x00	Legacy	Recommended Compatibility mode for legacy nodes. The two provider mode for the RT_CLASS_3 startup is required
0x01	Advanced	Mandatory for this document.

Bit 31: ARProperties.PullModuleAlarmAllowed

This field shall be set according to Table 453.

Table 453 – ARProperties.PullModuleAlarmAllowed

Value (hexadecimal)	Meaning	Use
0x00	The AlarmType(=Pull) shall signal pulling of submodule and module. The subslot number zero shall code 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 code 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 454.

Table 454 – IOCRProperties.RTClass

Value (hexadecimal)	Meaning	Use
0x00	Reserved	—
0x01	Use FrameID range 7 for the IOCRs See Table 45	Data-RTC-PDU RT_CLASS_1(legacy) should be supported by IO controller and IO supervisor together with ARProperties.StartupMode:= Legacy. It should not be generated by an IO device.
0x02	Use FrameID range 6 for the IOCRs See Table 44	Data-RTC-PDU
0x03	Use FrameID range 3 for the IOCRs See Table 41	Data-RTC-PDU
0x04	Use FrameID range 7 for the IOCRs See Table 45	UDP-RTC-PDU
0x05 – 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 455.

Table 455 – IOCRTagHeader.IOCRVLANID

Value (hexadecimal)	Meaning
0x000	Default No VLAN
0x001 – 0xFFFF	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 456.

Table 456 – IOCRTagHeader.IOUserPriority

Value (hexadecimal)	Meaning
0x00 – 0x05	Reserved
0x06	IO CR Priority
0x07	Reserved

5.2.4.14 Coding of the field IOCRTType

This field shall be coded as data type Unsigned16 with the values according to Table 457.

Table 457 – IOCRTType

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 458 and Table 459.

Table 458 – 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 459 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess==1 or ARProperties.StartupMode==1

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 from the CMDEV for the calculation of the CMInitiatorActivityTimeout according to Formula (50).

$$\text{CMInitiatorActivityTimeout} = \text{CMInitiatorActivityTimeoutFactor} \times 100 \text{ ms} \quad (50)$$

where

CMInitiatorActivityTimeout is the CM initiator activity timeout

CMInitiatorActivityTimeoutFactor is the CM initiator activity timeout factor

This field shall be used from the CMCTL for the calculation of the CMInitiatorTriggerTimeoutFactor which is used for the AR establishing with the values according to Formula (51) and Table 460.

$$\text{CMInitiatorTriggerTimeoutFactor} = \text{CMInitiatorActivityTimeoutFactor} \text{ DIV } 3 \quad (51)$$

where

CMInitiatorActivityTimeoutFactor is the CM initiator activity timeout factor

CMInitiatorTriggerTimeoutFactor is the CM initiator trigger timeout factor

Table 460 – 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 (52).

$$\text{CMInitiatorTriggerTimeout} = \text{CMInitiatorTriggerTimeoutFactor} \times 100 \text{ ms} \quad (52)$$

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 according to the rules of 4.3.1.4.15.1.

5.2.4.18 Coding of the field CMResponderStationName

This field shall be coded according to the rules of 4.3.1.4.15.1.

5.2.4.19 Coding of the field ParameterServerStationName

This field shall be coded according to the rules of 4.3.1.4.15.1.

5.2.4.20 Coding of the field ProviderStationName

This field shall be coded according to the rules of 4.3.1.4.15.1.

5.2.4.21 Coding of the field IODataObjectFrameOffset

This field shall be coded as data type Unsigned16 according to Table 461. 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 461 – 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 462.

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 462 – 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 463.

Table 463 – 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 464.

Table 464 – 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 465.

Table 465 – 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 466.

Table 466 – 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 467.

Table 467 – 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 468.

Table 468 – AlarmCRTagHeaderHigh.AlarmCRVLANID

Value (hexadecimal)	Meaning
0x000	Default No VLAN
0x001 – 0xFFFF	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 469.

Table 469 – AlarmCRTagHeaderHigh.AlarmUserPriority

Value (hexadecimal)	Meaning
0x00 – 0x05	Reserved
0x06	Alarm CR Priority High
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 470.

Table 470 – AlarmCRTagHeaderLow.AlarmCRVLANID

Value (hexadecimal)	Meaning
0x000	Default No VLAN
0x001 – 0xFFFF	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 471.

Table 471 – AlarmCRTagHeaderLow.AlarmUserPriority

Value (hexadecimal)	Meaning
0x00 – 0x04	Reserved
0x05	Alarm CR Priority Low
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 472.

Table 472 – 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 473.

Table 473 – 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 474.
The time base is 100 ms.

The RTATimeout shall be calculated according to Formula (53).

$$\text{RTATimeout} = \text{RTATimeoutFactor} \times 100 \text{ ms} \quad (53)$$

where

RTATimeout is the RTA timeout
RTATimeoutFactor is the RTA timeout factor

Table 474 – 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 475.

Table 475 – RTARetries

Value (hexadecimal)	Meaning	Meaning
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 containing the following elements:

- Data1 as Unsigned32 (0xDEA00000)
- Data2 as Unsigned16 (0x6C97)
- Data3 as Unsigned16 (0x11D1)
- Data4 as array of two Unsigned8, Octet 1 (0x82) to Octet 2 (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 NDREPMAP PDU. If it is part of the NDRDataxxx PDUs then only Big Endian Format is used.

The value of the field PROFINETIOConstantValue shall be DEA00000-6C97-11D1-8271.

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 476.

Table 476 – 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 477. The time base is one second. The AddressResolutionInterval shall be calculated according to Formula (54).

Table 477 – AddressResolutionProperties.Factor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x0064	Mandatory
0x0065 – 0xFFFF	Optional

$$\text{AddressResolutionInterval} = \text{AddressResolutionProperties.Factor} \times 1 \text{ s} \quad (54)$$

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 478. The time base shall be 100 ms. The MCIMonitoringInterval shall be calculated according to Formula (55).

Table 478 – MCITimeoutFactor

Value (hexadecimal)	Meaning
0x0000 – 0x0064	Mandatory
0x0065 – 0xFFFF	Reserved

$$\text{MCIMonitoringInterval} = \text{MCITimeoutFactor} \times 100 \text{ ms} \quad (55)$$

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 shall be coded as data type Unsigned8.

5.2.4.37.3 Coding of the field InstanceHigh

This field shall be coded as data type Unsigned8.

The value zero for both InstanceHigh and InstanceLow is recommended for single instance devices.

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 479.

Table 479 – 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, DeviceType, OrderID, HWRevision, SWRevisionPrefix, SWRevision, and IM_Serial_Number uniquely identify 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 479.

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 480.

Table 480 – VendorIDLow

Value (hexadecimal)	Meaning
0x00	The value zero for both VendorIDHigh and VendorIDLow is reserved for IEC 61158-6-3 devices.
0x01 – 0xFF	Administrative number

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 481.

Table 481 – VendorIDHigh

Value (hexadecimal)	Meaning
0x00	The value zero for both VendorIDHigh and VendorIDLow is reserved for IEC 61158-6-3 devices.
0x01 – 0xEF	Administrative number
0xFF	The value 0xFF is reserved for profiles.

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 482.

Table 482 – 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 483.

Table 483 – SubmoduleIdentNumber

Value (hexadecimal)	Meaning
0x00000000	Used in conjunction with – ModuleDiffBlock together with SubmoduleState.IdentInfo:= NoSubmodule – function 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 administrative uniquely assigned 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 484, Table 485, Table 486, and Table 487.

Table 484 – 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 values which identifies an ARs. See Table 485 for IOCARSR

Table 485 – 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: 1 st AR of an ARset 02: 2 nd AR of an ARset 03: 3 rd AR of an ARset 04: 4 th 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 486 – 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 487. A MD5 hash, calculated over the content of the IODConnect.req without the IO controller dependant parts, may be used.

Table 487 – 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 488.

Table 488 – 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 489.

Table 489 – 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 490, Table 491 and Table 492.

Table 490 – ControlBlockProperties in conjunction with ControlCommand.ApplicationReady with ARProperties.StartupMode==1

Bitposition	Value (hexadecimal)	Meaning
Bit 0 – 15	—	Shall be set according to 3.4.2.2.

Table 491 – ControlBlockProperties in conjunction with ControlCommand.ApplicationReady with ARProperties.StartupMode==0

Bitposition	Value (hexadecimal)	Meaning
Bit 0	0x00	Wait for explicit ControlCommand.ReadyForCompanion
	0x01	Implicit ControlCommand.ReadyForCompanion
Bit 1	0x00	Wait for explicit ControlCommand.ReadyForRT_CLASS_3
	0x01	Implicit ControlCommand.ReadyForRT_CLASS_3
Bit 2 – 15	—	Shall be set according to 3.4.2.2.

Table 492 – 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 493.

Table 493 – 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 494.

Table 494 – 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 495.

Table 495 – 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 496.

Table 496 – 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 497.

Table 497 – 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 498.

Table 498 – 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 499.

Table 499 – 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 500.

Table 500 – 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 501.

NOTE The field DataLength is in minimum the sum of all DataItems. It can be larger.

Table 501 – Values of DataLength

Value (decimal)	Domain	Meaning
40 – 1 440	RT_CLASS_1, RT_CLASS_2 and RT_CLASS_3	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 values shall be set to zero.

5.2.4.55 Coding of the field Padding

This field shall be coded as data type Unsigned8. The values shall be set to zero.

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 values shall be set to zero.

5.2.4.58 Coding of the field SendClockFactor

This field shall be coded as data type Unsigned16 according to Table 502. The time base is 31,25 µs.

Table 502 – Values of SendClockFactor

Value (decimal)	LengthOfPeriod	Meaning
0	—	Reserved
1	31,25 µs	Optional
2	62,5 µs	Optional
3	93,75 µs	Optional
4	125 µs	Optional
5 – 7	—	Optional
8	250 µs	Optional
9 – 15	—	Optional
16	500 µs	Optional
17 – 31	—	Optional
32	1 ms	Mandatory
33 – 63	—	Optional
64	2 ms	Optional
65 – 127	—	Optional
128	4 ms	Optional
129 – 65 535	—	Reserved

The LengthOfPeriod corresponds with a value range from 31,25 µs to 4 000 µs.

A maximum Ethernet frame contains 1 518 or 1 522 octets. To transfer this kind of frame at a data rate of 100 Mbit/s the LengthOfPeriod should be greater than 122,24 µs and at a data rate of 1 Gbit/s the LengthOfPeriod should be greater than 12,224 µs.

The value of the LengthOfPeriod shall be calculated according to Formula (56).

$$\text{LengthOfPeriod} = \text{SendClockFactor} \times 31,25 \mu\text{s} \quad (56)$$

where

LengthOfPeriod is the length of the period
SendClockFactor 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.

5.2.4.59 Coding of the field ReductionRatio

This field shall be coded as data type Unsigned16 with the values according to Table 503, Table 504, Table 505, Table 506, and Table 507.

The allowed value shall be calculated in conjunction with Table 502 according Formula (57).

Table 503 – Values of ReductionRatio for RT_CLASS_1 and RT_CLASS_2

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 forth send clock
8	Mandatory	Transmit and receive every eighth send clock
16	Mandatory	Transmit and receive every sixteenth send clock
32	Mandatory	Transmit and receive every 32 send clock
64	Mandatory	Transmit and receive every 64 send clock
128	Mandatory	Transmit and receive every 128 send clock
256	Mandatory	Transmit and receive every 256 send clock
512	Mandatory	Transmit and receive every 512 send clock
513 – 65 535	Reserved	—
other	Optional	Not intended to be used

Table 504 – 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 forth 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 505 – Values of ReductionRatio for RT_CLASS_3 and SendClockFactor < 8

Value (decimal)	Meaning	Use
1	Mandatory	Transmit and receive every send clock
17 – 65 535	Reserved	—
other	Optional	Not intended to be used

**Table 506 – Values of ReductionRatio in conjunction with a non power of 2
SendClockFactor**

Value (decimal)	Meaning	Use
1	Optional	Transmit and receive every send clock
17 – 65 535	Reserved	—
other	Optional	Not intended to be used

Table 507 – 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 forth send clock
8	Optional	Transmit and receive every eighth send clock
16	Optional	Transmit and receive every sixteenth send clock
32	Optional	Transmit and receive every 32 send clock
64	Optional	Transmit and receive every 64 send clock
128	Mandatory	Transmit and receive every 128 send clock
256	Mandatory	Transmit and receive every 256 send clock
512	Mandatory	Transmit and receive every 512 send clock
1 024	Mandatory	Transmit and receive every 1 024 send clock
2 048	Mandatory	Transmit and receive every 2 048 send clock
4 096	Mandatory	Transmit and receive every 4 096 send clock
8 192	Mandatory	Transmit and receive every 8 192 send clock
16 384	Mandatory	Transmit and receive every 16 384 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 508. It shall be less or equal to the related ReductionRatio.

Table 508 – Values of Phase

Value (decimal)	Meaning	Use
0	Reserved	—
1 – 16 384	Mandatory	But 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 509.

Table 509 – Values of Sequence

Value (hexadecimal)	Meaning	Use
0	Mandatory, sequence undefined	The PPMs uses a random sequence for each phase
0x0001 – 0xFFFF	Optional, sequence defined	The PPMs uses the given sequence for each phase

NOTE For RT_CLASS_3 is a given sequence 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 with ReductionRatio of the monitored consumer.

The DataHoldTime shall be calculated according to Formula (57).

$$\text{DataHoldTime} = \text{DataHoldFactor} \times \text{SendClockFactor} \times \text{ReductionRatio} \times 31,25 \mu\text{s} \quad (57)$$

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 equal or less than 1,92 s.

The value range for UDP-RTC-PDUs is from 0x0003 to 0xF000. The DataHoldTime shall equal or less than 61,44 s.

For a frame, the usage of the DataHoldFactor is defined in Table 510.

Table 510 – 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	—

For a Subframe, the usage of the DataHoldFactor is defined in Table 511.

Table 511 – 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 and set according to Table 512, Figure 112 and Formula (58). The time base is 1 ns.

NOTE The minimal planned gap between two RT_CLASS_3 frames is 1 120 ns.

Table 512 – Values of FrameSendOffset

Value (decimal)	Meaning	Use
0 – 0x00001388	Relative send offset to the start of the related cycle	Optional
0x00001389 – 0x003B28FF	Relative send offset to the start of the related cycle	Mandatory for RT_CLASS_3 Optional for RT_CLASS_1, RT_CLASS_2 and RT_CLASS_UDP
0x003B2900 – 0x003D08FF	Relative send offset to the start of the related cycle Only useable if the frame size fulfill Formula (58)	Mandatory for RT_CLASS_3 Optional for RT_CLASS_1, RT_CLASS_2 and RT_CLASS_UDP
0x003D0900 – 0xFFFFFFFF	Reserved	Not used
0xFFFFFFFF	Best effort Provider protocol machines sends the frame as soon as possible	Mandatory Not used for PDIRFrameData

$$\text{FrameSendOffset}[\text{Frame}] + \text{Duration}[\text{Frame}] < \text{SendClockFactor} \times 31,25 \mu\text{s} \quad (58)$$

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

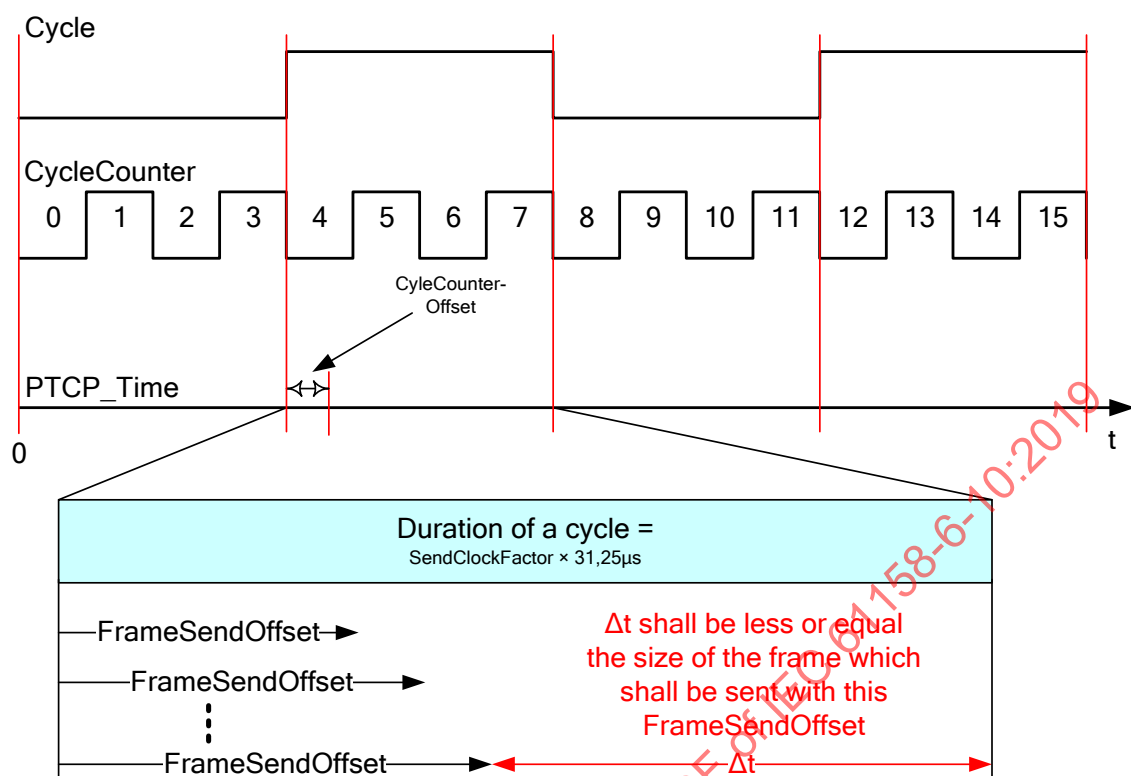


Figure 112 – FrameSendOffset vs. duration of a cycle

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 513.

Table 513 – 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 by 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 520 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 514.

Table 514 – SubmoduleState.AddInfo

Value (hexadecimal)	Meaning	Description
0x00	None	—
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 515. It summarizes the Advice information of the Diagnosis ASE for the reporting submodule.

Table 515 – 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 516. It summarizes the MaintenanceRequired information of the Diagnosis ASE for the reporting submodule.

Table 516 – 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 517. It summarizes the MaintenanceDemanded information of the Diagnosis ASE for the reporting submodule.

Table 517 – 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 518. It summarizes the Fault information of the Diagnosis ASE for the reporting submodule.

Table 518 – 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 519.

Table 519 – 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 520.

Table 520 – 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 521.

Table 521 – 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, .ARInfo and .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 522.

Table 522 – 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 follows

Bit 2: SubmoduleProperties.SharedInput

This field shall be set according to Table 523.

Table 523 – 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 524.

Table 524 – SubmoduleProperties.ReduceInputSubmoduleDataLength

Value (hexadecimal)	Meaning	Use
0x00	Expected	Use expected input SubmoduleDataLength for I-CR
0x01	Zero	Reduce input SubmoduleDataLength to zero for I-CR. Shall only be set if SubmoduleProperties.Type == INPUT or SubmoduleProperties.Type == INPUT_OUTP UT is used.

Bit 4: SubmoduleProperties.ReduceOutputSubmoduleDataLength

This field shall be set according to Table 525.

Table 525 – SubmoduleProperties.ReduceOutputSubmoduleDataLength

Value (hexadecimal)	Meaning	Use
0x00	Expected	Use expected output SubmoduleDataLength for O-CR
0x01	Zero	Reduce output SubmoduleDataLength to zero for O-CR. Shall only be set if SubmoduleProperties.Type == OUTPUT or SubmoduleProperties.Type == INPUT_OUTP UT is used.

Bit 5: SubmoduleProperties.DiscardIOXS

This field shall be set according to Table 526.

Table 526 – 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 527.

Table 527 – 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 528.

Table 528 – 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 529.

Table 529 – InitiatorUDPRTPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Optional Port may 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 530.

Table 530 – ResponderUDPRTPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Optional Port may 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 531.

Table 531 – InitiatorRPCServerPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Reserved
0x0400 – 0xBFFF	IANA_DEFINED_PORT	Useable
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 532.

Table 532 – ResponderRPCServerPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Reserved
0x0400 – 0xBFFF	IANA_DEFINED_PORT	Useable
0xC000 – 0xFFFF	IANA_FREE_PORT is used as dynamic and/or private port	Recommended

5.2.4.74 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.

5.2.4.75 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.

5.2.4.76 Coding of the field MaxAlarmDataLength

This field shall be coded as data type Unsigned16 with values according to Table 533.

Table 533 – MaxAlarmDataLength

Value (hexadecimal)	Meaning
0x00C8	Mandatory Maximal size of the AlarmNotification-PDU accepted by the AlarmEndpoint.
0x00C9 – 0x0598	Optional Maximal size of the AlarmNotification-PDU accepted by the AlarmEndpoint.
Other	Reserved

An AlarmEndpoint shall not create larger AlarmNotification-PDUs than negotiated during AR establishment.

5.2.4.77 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 534 and Table 535.

Table 534 – 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 535 – 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 536.

Table 536 – 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 537.

Table 537 – 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 value 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 value 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 TargetARUID == NIL and ARUID == 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 538.

Table 538 – 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 only used with ErrorDecode=PNIORW
0xDF	IODWriteRes	Within the CL-RPC-PDU only used with ErrorDecode=PNIORW
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 539.

Table 539 – Values of ErrorDecode

Value (hexadecimal)	Meaning	Use
0x00	Reserved	See Table 538
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 indicate that the parameters ErrorCode1 shall be set according to Table 540. The ErrorCode1 consists of ErrorClass and ErrorDecode. See also ErrorCode Hierarchy in IEC 61151-5-10.

Table 540 – 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 constrain conflict 1 = write constrain 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 541.

Table 541 – 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 542 shows the values for ErrorCode1 together with ErrorDecode:= PNIO.

Table 542 – 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 – 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
0x32	Response Parameter Error Faulty ARBlockRes

ErrorCode1 (hexadecimal)	Meaning
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 – 0x63	Reserved
0x64	CTLSM
0x65	CTLRDI
0x66	CTLRDR
0x67	CTLWRI
0x68	CTLWRR
0x69	CTLIO
0x6A	CTLSU
0x6B	CTLRPC
0x6C	CTLPBE
0x6D	CTLSRL

ErrorCode1 (hexadecimal)	Meaning
0x6E – 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 – 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 543, Table 544, Table 545, Table 546, Table 547, Table 548, Table 549 and the definitions of 4.10.3.4.6. For usage, see 5.2.31.2 containing the PDU checking rules.

Table 543 – 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
Read Write Record Parameter Error	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length

Meaning	Value ErrorCode2 (hexadecimal)	Meaning/Usage
Faulty Record	...	...
	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 ARFSUBBlock	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
IODControl Parameter Error Faulty ControlBlockConnect	0x00	Error in Parameter BlockType
	0x01	Error in Parameter Length
	...	...
	0x09	Error in Parameter ControlBlockProperties
	0x0A – 0xFF	Reserved
IODControl Parameter Error Faulty ControlBlockPlug	0x00	Error in Parameter BlockType
	...	...
	0x09	Error in Parameter ControlBlockProperties
	0x0A – 0xFF	Reserved
IOXControl Parameter Error Faulty ControlBlockConnect after	0x00	Error in Parameter BlockType
	...	...

Meaning	Value ErrorCode2 (hexadecimal)	Meaning/Usage
a connection establishment	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
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	0x00	Error in Parameter BlockType

Meaning	Value ErrorCode2 (hexadecimal)	Meaning/Usage
Faulty ModuleDiffBlock	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 544 – 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 545 – 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 288 and Table 289. (CLRPC_ERR_REJECTED)
	0x02	Server had a fault while executing the call. For further details see Table 288 and Table 289. (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
APMS	0x00	Invalid state
	0x01	LMPM signaled an error

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
	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

Table 546 – 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
CTLISM	0x00	Invalid state
	0x01	CTLISM signaled an error

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
	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
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

**Table 547 – 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 – 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
CMSU	0x00	Invalid state
	0x01	AR add provider or consumer failed
	0x02	AR alarm-open failed
	0x03	AR alarm-send

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
	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

**Table 548 – 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
	0x01	Error within the coordination of sequence numbers (RTA_ERR_CODE_SEQ) error
	0x02	Instance closed (RTA_ERR_ABORT)
	0x03	AR out of memory (RTA_ERR_ABORT)
	0x04	AR add provider or consumer failed (RTA_ERR_ABORT)
	0x05	AR consumer DHT expired (RTA_ERR_ABORT)
	0x06	AR cmi timeout (RTA_ERR_ABORT)
	0x07	AR alarm-open failed (RTA_ERR_ABORT)
	0x08	AR alarm-send.cnf (-) (RTA_ERR_ABORT)
	0x09	AR alarm-ack-send.cnf (-) (RTA_ERR_ABORT)
	0x0A	AR alarm data too long (RTA_ERR_ABORT)

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
	0x0B	AR alarm.ind (err) (RTA_ERR_ABORT)
	0x0C	AR rpc-client call.cnf (-) (RTA_ERR_ABORT)
	0x0D	AR abort.req (RTA_ERR_ABORT)
	0x0E	AR re-run aborts existing (RTA_ERR_ABORT)
	0x0F	AR release.ind received (RTA_ERR_ABORT)
	0x10	AR device deactivated (RTA_ERR_ABORT)
	0x11	AR removed (RTA_ERR_ABORT)
	0x12	AR protocol violation (RTA_ERR_ABORT)
	0x13	AR name resolution error (RTA_ERR_ABORT)
	0x14	AR RPC-Bind error (RTA_ERR_ABORT)
	0x15	AR RPC-Connect error (RTA_ERR_ABORT)
	0x16	AR RPC-Read error (RTA_ERR_ABORT)
	0x17	AR RPC-Write error (RTA_ERR_ABORT)
	0x18	AR RPC-Control error (RTA_ERR_ABORT)
	0x19	AR forbidden pull or plug after check.rsp and before in-data.ind (RTA_ERR_ABORT)
	0x1A	AR AP removed (RTA_ERR_ABORT)
	0x1B	AR link down (RTA_ERR_ABORT)
	0x1C	AR could not register multicast-mac address (RTA_ERR_ABORT)
	0x1D	Not synchronized (cannot start companion-ar) (RTA_ERR_ABORT)
	0x1E	Wrong topology (cannot start companion-ar) (RTA_ERR_ABORT)

ErrorCode1	Value ErrorCode2 (hexadecimal)	Meaning/Usage
	0x1F	DCP, station-name changed (RTA_ERR_ABORT)
	0x20	DCP, reset to factory or factory reset (RTA_ERR_ABORT)
	0x21	Can't start companion-AR because a 0xLIPP submodule in the first AR... (RTA_ERR_ABORT)
	0x22	No IRDATA record yet (RTA_ERR_ABORT)
	0x23	Pdev, owner is leaving (RTA_ERROR_ABORT)
	0x24	Pdev, no port offers required speed/duplexity (RTA_ERROR_ABORT)
	0x25	IP-Suite [of the IOC] changed by means of DCP_set (IPParameter) or local engineering (RTA_ERROR_ABORT)
	0x26	IOCARSR RDHT expired (RTA_ERROR_ABORT)
	0x27	IOCARSR Pdev, parametrization impossible (RTA_ERROR_ABORT)
	0x28	Remote application ready timeout expired (RTA_ERROR_ABORT)
	0x29	IOCARSR Redundant interface lost or access to the peripherals impossible (RTA_ERROR_ABORT)
	0x2A	IOCARSR MTOT expired (RTA_ERROR_ABORT)
	0x2B	IOCARSR AR protocol violation (RTA_ERROR_ABORT)
	0x2C	Pdev, plug port without CombinedObjectContainer (RTA_ERROR_ABORT)
	0x2D – 0xC8	Reserved
	0xC9 – 0xFF	Manufacturer specific
User specific	0x00 – 0xFE	User specific
	0xFF	Recommended for "User abort" without further detail

Table 549 – 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 550 and Table 551.

Table 550 – Coding of ErrorCode1 for ErrorDecode with the value ManufacturerSpecific

ErrorCode1 (hexadecimal)	Meaning	Usage
0x00 – 0xFF	Manufacturer specific	—

Table 551 – 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 552.

Table 552 – Visible characters

VisibleString (hexadecimal)	Usage
0x00 – 0x1F	Control characters
0x20 – 0x7E	Visible characters
0x7F	Control character

5.2.7.2 Reset behavior

Either Table 553 or Table 554 show the behavior after FactoryReset and ResetToFactory depending on the source of the named fields. Table 555 covers the special case usage of I&M4 together with functional safety submodules.

**Table 553 – FactoryReset / ResetToFactory behavior
(legacy from IEC 61158-x-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 554 – FactoryReset / ResetToFactory behavior
(default without IEC 61158-x-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 555 – FactoryReset / ResetToFactory behavior if used in conjunction 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 manufacturer specific according to 5.2.7.1.

It uniquely identifies one produced entity and is thus an extension of

- VendorID, DeviceID, ModuleIdentNumber, and SubmoduleIdentNumber and/or
- VendorID, DeviceID, DeviceType, OrderID, HWRevision, SWRevisionPrefix, and SWRevision

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 556.

Table 556 – 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 557.

Table 557 – 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 558.

Table 558 – 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 559.

Table 559 – 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 560.

Table 560 – 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 561. This field and the field API (see 5.2.3.1) are related.

Table 561 – 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 562 and Table 563.

Table 562 – 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 563 – 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
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

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 564.

Table 564 – 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 565.

Table 565 – 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 566.

Table 566 – 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 566.

Bit 3: IM_Supported.I&M3

This field shall be set according to Table 566.

Bit 4: IM_Supported.I&M4

This field shall be set according to Table 566.

If used for functional safety, additional rules may apply.

Bit 5: IM_Supported.I&M5

This field shall be set according to Table 566.

Bit 6: IM_Supported.I&M6

This field shall be set according to Table 566.

Bit 7: IM_Supported.I&M7

This field shall be set according to Table 566.

Bit 8: IM_Supported.I&M8

This field shall be set according to Table 566.

Bit 9: IM_Supported.I&M9

This field shall be set according to Table 566.

Bit 10: IM_Supported.I&M10

This field shall be set according to Table 566.

Bit 11: IM_Supported.I&M11

This field shall be set according to Table 566.

Bit 12: IM_Supported.I&M12

This field shall be set according to Table 566.

Bit 13: IM_Supported.I&M13

This field shall be set according to Table 566.

Bit 14: IM_Supported.I&M14

This field shall be set according to Table 566.

Bit 15: IM_Supported.I&M15

This field shall be set according to Table 566.

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 567 or Table 568.

The string format “YYYY-MM-DD'T'HH:MM'Z” or “YYYY-MM-DD'T'HH:MM'±HH:MM” is in accordance with the ISO 8601. 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 567 should be ‘Z’.

Table 567 – 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 568 – 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 569.

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 569 – 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 570.

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 570 – IM_OrderID

Meaning	Usage
Vendor specific defined Order Identification	—

5.2.7.21 Coding of the field `IM_UniquelIdentifier`

This field shall be coded as data type `UUID` with the values according to Table 571.

Table 571 – IM_UniquelIdentifier

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 PDUs

5.2.8.1 Coding of the field `UserStructureIdentifier`

This field shall be coded as data type `Unsigned16` with the values according to Table 572. This field identifies the structure of the field `Data` of the `AlarmNotification` and structure of the field `Data` of the alarm data.

Table 572 – 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
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	—

Value (hexadecimal)	Meaning	Usage
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
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 573, Table 574, Table 575, and Table 576.

Table 573 – 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	Line break	Line 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
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 less 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 574 – 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 from 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
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 575 – ChannelErrorType – range 3

Value (hexadecimal)	Meaning	Assigned text
0x0100 – 0x7FFF	Manufacturer specific	Manufacturer specific

Table 576 – 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
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 – 0x8FFF	Reserved	Unknown error
0x9000 – 0x9FFF	Reserved for profiles ^a	Profile specific
0xA000 – 0xFFFF	Reserved	Unknown error
^a Central administrative number to unambiguously distinguish between profiles		

5.2.8.3 Coding of the field ChannelNumber

This field shall be coded as data type Unsigned16 with the values according to Table 577.

Table 577 – 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 with the ChannelProperties.Direction.

In dependence with the field ChannelProperties.Accumulative the following meaning shall be applied:

- ChannelProperties.Accumulative == 1 then the field ChannelNumber shall contain the number of the smallest 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 578.

Table 578 – 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 below defined types are 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 579.

Table 579 – 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 580, Table 581 and Table 582.

The dependencies with elements of the field ChannelProperties are specified in Table 581 and the dependencies with Alarmnotification and RecordDataRead(DiagnosisData) in Table 582.

Table 580 – 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 113).		

Table 581 – 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 other remain	
Maintenance required	0x00	Reserved	ChannelDiagnosis, ExtChannelDiagnosis, QualifiedChannelDiagnosis
	0x01	MaintenanceRequired appears	
	0x02	MaintenanceRequired disappears	
	0x03	MaintenanceRequired disappears but other remain	
Maintenance demanded	0x00	Reserved	ChannelDiagnosis, ExtChannelDiagnosis, QualifiedChannelDiagnosis
	0x01	MaintenanceDemanded appears	
	0x02	MaintenanceDemanded disappears	
	0x03	MaintenanceDemanded disappears but other remain	
Qualified-Channel-Qualifier	0x00	Reserved	QualifiedChannelDiagnosis
	0x01	QualifiedDiagnosis appears	
	0x02	QualifiedDiagnosis disappears	
	0x03	QualifiedDiagnosis disappears but other remain	

^a Subsequent means, all ExtChannelErrorTypes for the delivered ChannelErrorType and also the ChannelErrorType itself disappears. Shall be used in AlarmNotification only.

Table 582 – 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 other remain	Alarmnotification
Maintenance required	0x00	Reserved	—
	0x01	MaintenanceRequired appears	Alarmnotification and Record-DataRead(DiagnosisData)
	0x02	MaintenanceRequired disappears	Alarmnotification
	0x03	MaintenanceRequired disappears but other remain	
Maintenance demanded	0x00	Reserved	—
	0x01	MaintenanceDemanded appears	Alarmnotification and Record-DataRead(DiagnosisData)
	0x02	MaintenanceDemanded disappears	Alarmnotification
	0x03	MaintenanceDemanded disappears but other remain	
Qualified diagnosis	0x00	Reserved	—
	0x01	QualifiedDiagnosis appears	Alarmnotification and Record-DataRead(DiagnosisData)
	0x02	QualifiedDiagnosis disappears	Alarmnotification
	0x03	QualifiedDiagnosis disappears but other remain	

^a Subsequent means, all ExtChannelErrorTypes for the delivered ChannelErrorType and also the ChannelErrorType itself disappears.

Bit 11 – 12: ChannelProperties.Specifier

This field shall be coded with the values according to Table 583. The field ChannelProperties.Specifier is related to the addressed channel and severity (fault or maintenance). The dependencies with elements of the field ChannelProperties are specified in Table 581.

Table 583 – ChannelProperties.Specifier

Value (hexadecimal)	Meaning	Usage
0x00	All subsequent disappears	See Table 581 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 other 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 584.

Table 584 – 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

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 585.

Table 585 – ExtChannelErrorType

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Definition is depending on the ChannelErrorType (see tables below).

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 587, Table 588, Table 589, Table 590, Table 591, Table 592, Table 593, Table 594, Table 595, Table 596, Table 597, Table 598, Table 599, Table 600 and Table 601.

Additionally, an ExtChannelErrorType value range is bound to the type of the ChannelErrorType according to Table 586.

Table 586 – Allowed combinations of ChannelErrorType, ExtChannelErrorType, and ExtChannelAddValue

ChannelErrorType	ExtChannelErrorType	ExtChannelAddValue	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

Table 587 – 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 588 – Additional ExtChannelErrorType for ChannelErrorType 0x0F and 0x10

Value (hexadecimal)	Meaning	Usage
0x8001	Parameter fault detail	Alarm/diagnosis
0x8002	Consistency fault detail	Alarm/diagnosis

Table 589 – 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	—

Table 590 – 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	—

Table 591 – 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 occurs, if the remote RTClass3_PortStatus.State is not RTCLASS3_RUN. This could also happen, if the remote IRDATAUUID is not equal to the local IRDATAUUID.

^b This event occurs, 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.

**Table 592 – 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 (59)
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 Formula (59).

$$\text{ExtChannelErrorType} = 0x800X + (\text{MRP\_Instance} \times 0x10) \quad (59)$$

where

ExtChannelErrorType is the ExtChannelErrorType used for the signaling of an event
 0x800X is the base value of the ExtChannelErrorType from Table 592
 MRP_Instance is the MRP_Instance of the MRP ring issuing the event
 0x10 is a constant value

**Table 593 – 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.

Table 594 – 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	—

Table 595 – 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 MLS update detected	Alarm/diagnosis
0x8003 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

Table 596 – 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	—

Table 597 – 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 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

**Table 598 – 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	—

**Table 599 – 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 (60)	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 Formula (60).

$$\text{ExtChannelErrorType} = 0x8000 + \text{FrameID} \quad (60)$$

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

**Table 600 – 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 (61)	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 least SlotNumber/SubslotNumber combination.

The used ExtChannelErrorType shall be calculated according Formula (61).

$$\text{ExtChannelErrorType} = 0x8000 + \text{FrameID} \quad (61)$$

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

Table 601 – 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	—

5.2.8.6 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 602.

Table 602 – 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 603 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0 – 0x7FFF.

Table 603 – 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 604, Table 605, and Table 606 if the field ExtChannelErrorType contains the value 0x8001 and the field ChannelErrorType contains the value 0 – 0xFF.

Table 604 – 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 605 – 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 606 – 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”: Additional 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 607, Table 608, and Table 606 if the field ExtChannelErrorType contains the value 0x8002 and the field ChannelErrorType contains the value 0 – 0xFF.

Table 607 – 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 608 – 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 609 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0x8007.

Table 609 – Values for “Fiber optic mismatch” – “Power Budget”

Value (hexadecimal)	Meaning
0x00 – 0x3E7	PowerBudget in 0,1 dB steps [0..99,9 dB]
0x03E8 – 0xFFFFFFFF	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 610 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0x8008.

Table 610 – 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 611 if the field ExtChannelErrorType contains the value 0x8007 and the field ChannelErrorType contains the value 0x8001.

Table 611 – 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 612 if the field ExtChannelErrorType contains the value 0x8003 and the field ChannelErrorType contains the value 0x800D.

Table 612 – 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 one.	Inconsistency
0x02	MultipleInterfaceMode.NameOfDevice for this interface is set to one 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 613 if the field ExtChannelErrorType contains the value 0x8004 and the field ChannelErrorType contains the value 0x800D.

Table 613 – 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 614. There shall only none or one bit be set 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 614 – 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. At least one of the bits below shall be set to one in order to convey this alarm block according to Table 615. Otherwise this block shall be omitted.

Table 615 – 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 614.			
^b This bit summarizes the MaintenanceDemanded information in all codings. See Table 614.			

The classification of fault, maintenance and normal operation according to their severity is shown in Figure 113.

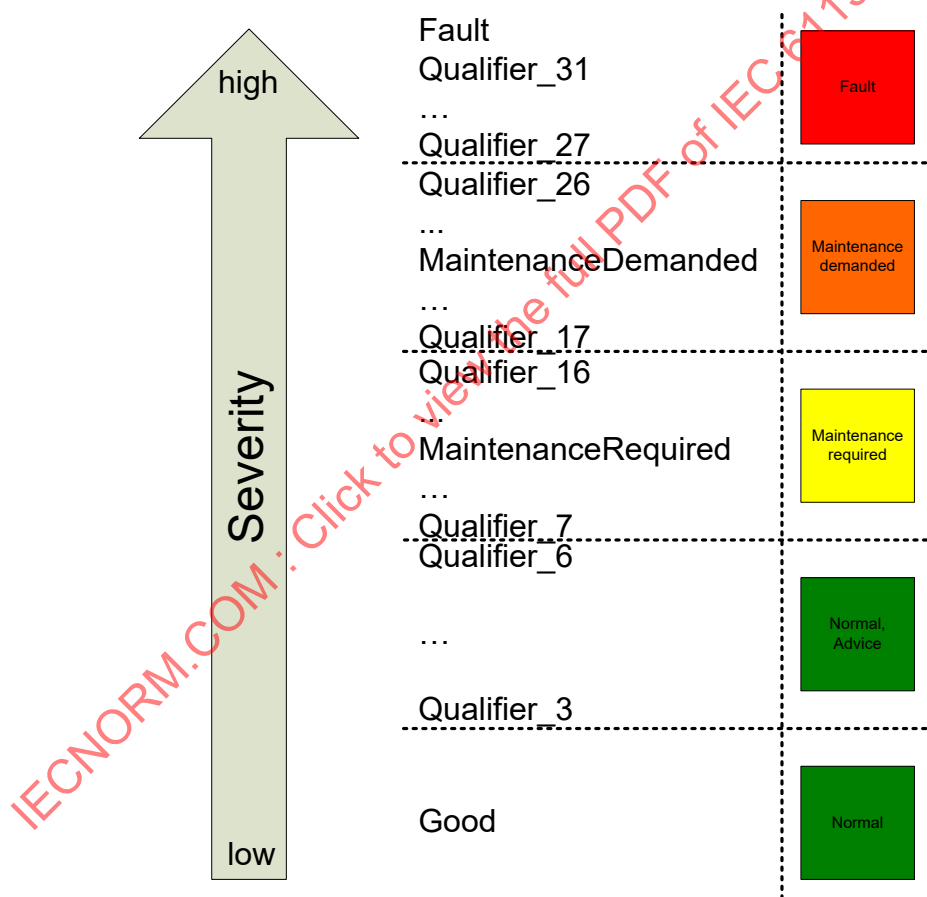


Figure 113 – Severity classification of fault, maintenance and normal

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 over 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 616.

Table 616 – 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 617.

Table 617 – 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 or equal 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 618.

Table 618 – 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 619.

Table 619 – 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 620.

Table 620 – 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 621.

Table 621 – Total_iPar_Size

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Depending on the used application protocol.

5.2.11 Coding section related to Physical Device Interface Data**5.2.11.1 Void****5.2.11.2 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 622.

Table 622 – MultipleInterfaceMode.NameOfDevice

Value (hexadecimal)	Meaning
0x00	Default The field LLDPChassis contains the NameOfStation and the field PortID of LLDP contains the name of the port.
0x01	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 Coding section related to Physical Device Port Data**5.2.12.1 Coding of the field OwnPortName**

This field shall be coded as data type OctetString[] according to 4.3.1.4.16.

5.2.12.2 Coding of the field LengthOwnPortName

This field shall be coded as data type Unsigned8 according to 4.3.1.4.16.

5.2.12.3 Coding of the field NumberOfPeers

This field shall be coded as data type Unsigned8 according to Table 623 and Table 624.

Table 623 – NumberOfPeers in conjunction with PDPortDataCheck

Value (hexadecimal)	Meaning
0x01	One neighbor shall be checked
Other	Reserved

Table 624 – 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

5.2.12.4 Coding of the field LengthPeerPortName

This field shall be coded as data type Unsigned8.

5.2.12.5 Coding of the field PeerPortName

5.2.12.5.1 Usage in conjunction with PDPortDataCheck.CheckPeers

This field shall be coded according to 4.3.1.4.16.

5.2.12.5.2 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 296, 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.

5.2.12.6 Coding of the field LengthPeerStationName

This field shall be coded as data type Unsigned8.

5.2.12.7 Coding of the field PeerStationName

5.2.12.7.1 General

This field shall be coded as data type OctetString[] with 1 to 255 octets.

5.2.12.7.2 Usage in conjunction with PDPortDataCheck.CheckPeers

This field shall be coded according to 4.3.1.4.15.

5.2.12.7.3 Usage in conjunction with PDPortDataReal or PDPortDataRealExtended

This field shall be coded according to 4.3.1.4.15.

If the received LLDP_PortID satisfies the coding according to Table 296, 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.

5.2.12.8 Coding of the field LengthOwnStationName

This field shall be coded as data type Unsigned8 according to 4.3.1.4.15.

5.2.12.9 Coding of the field OwnStationName

This field shall be coded as data type OctetString[] according to 4.3.1.4.15.

5.2.12.10 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 625, Table 626, Figure 41 and Formula (29).

Table 625 – LineDelay.Value with LineDelay.FormatIndicator == 0

Value (hexadecimal)	Meaning
0x00000000	Line delay and cable delay unknown
0x00000001 – 0x7FFFFFFF	Line delay in nanoseconds

Table 626 – 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 627.

Table 627 – 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 Shall be used in conjunction with PDPortDataReal or PDPortDataRealExtended

5.2.12.11 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 802.

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

5.2.13 Coding section related to Physical Device IR Data**5.2.13.1 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 628.

Table 628 – RxPort

Value (hexadecimal)	Meaning
0x00	Local interface
0x01	Port 1
0x02	Port 2
...	...
0xFF	Port 255

5.2.13.2 Coding of the field NumberOfTxPortGroups

This field shall be coded as data type Unsigned8 and shall be set according to Table 629. This field shall only count the succeeding TxPortGroupArray entries.

Table 629 – 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

5.2.13.3 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 octet corresponds to the number of ports within a device and shall be calculated as follows

$$N = M_{\text{highest}} \text{ DIV } 8 + 1 \quad (62)$$

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 a octet

$$G = m \text{ DIV } 8 + 1 \quad (63)$$

where

G	is the number of the TxPortGroup octet containing the TxPort
m	is the number of the current TxPort with $1 \leq m \leq M$
8	is the bit size of a octet

$$B = m \text{ MOD } 8 \quad (64)$$

where

B	is the number of the bit in the TxPortGroup octet containing the TxPort
m	is the number of the current TxPort with $1 \leq m \leq M$
8	is the bit size of a 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) may not contain all 8 TxPort entries. If $M_{\text{highest}} \text{ MOD } 8 \neq 7$ 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 630 if the TxPort with the number that meets $m \text{ MOD } 8 = 0$ is present. A padding bit shall be used if no TxPort is present.

Table 630 – 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 630 if the TxPort with the number that meets $m \bmod 8 = 1$ 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 630 if the TxPort with the number that meets $m \bmod 8 = 2$ 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 630 if the TxPort with the number that meets $m \bmod 8 = 3$ 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 630 if the TxPort with the number that meets $m \bmod 8 = 4$ 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 630 if the TxPort with the number that meets $m \bmod 8 = 5$ 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 630 if the TxPort with the number that meets $m \bmod 8 = 6$ 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 630 if the TxPort with the number that meets $m \bmod 8 = 7$ is present. A padding bit shall be used if no TxPort is present.

5.2.13.4 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 631 and Table 632.

**Table 631 – FrameDetails.SyncFrame in conjunction with
FrameDataProperties.ForwardingMode=="Absolute mode"**

Value (hexadecimal)	Meaning	Description
0x00	No sync frame	Mandatory
0x01	Primary sync frame	Legacy Primary and secondary sync frame use the same FrameID. At one time only one frame shall be valid.
0x02	Secondary sync frame	
0x03	Reserved	—

**Table 632 – 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 633.

Table 633 – 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 634.

Table 634 – 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

5.2.13.5 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 635.

Table 635 – FrameDataProperties.ForwardingMode

Value (hexadecimal)	Meaning
0x00	Absolute mode The FrameSendOffset contains the sending time for a forwarded frame
0x01	Relative mode The FrameSendOffset shall contain the sending time of a forwarded frame or the “Best effort” value.

Bit 1 – 2: FrameDataProperties.FastForwardingMulticastMACAdd

This field shall be set according to Table 636.

Table 636 – 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 637.

Table 637 – 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.

5.2.13.6 Coding of the field AdjustProperties

This field shall be coded as data type Unsigned16 with the value zero.

5.2.13.7 Coding of the field MAUType

This field shall be coded as data type Unsigned16 with the values according to IETF RFC 4836, Table 638, Table 639, and Table 640.

Table 638 – MAUType

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved for MediaType "Copper cable" and "Fiber optic cable"	Default for LinkState.Link != "UP"
	Used for MediaType "Radio communication"	Default for radio communication
0x0005	10BaseT	PDPortDataReal, PDPortDataRealExtended
0x000A	10BaseTXHD	PDPortDataReal, PDPortDataRealExtended
0x000B	10BaseTXFD	PDPortDataReal, PDPortDataRealExtended
0x000C	10BaseFLHD	PDPortDataReal, PDPortDataRealExtended
0x000D	10BaseFLFD	PDPortDataReal, PDPortDataRealExtended
0x000F	100BaseTXHD	PDPortDataReal, PDPortDataRealExtended
0x0010	Default (MediaType Copper) 100BaseTXFD	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0011	100BaseFXHD	PDPortDataReal, PDPortDataRealExtended

Value (hexadecimal)	Meaning	Usage
0x0012	Default (Media Type Fiber optic) 100BaseFXFD	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0015	1000BaseXHD	PDPortDataReal, PDPortDataRealExtended
0x0016	1000BaseXFD	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0017	1000BaseLXHD	PDPortDataReal
0x0018	1000BaseLXFD	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0019	1000BaseSXHD	PDPortDataReal, PDPortDataRealExtended
0x001A	1000BaseSXFD	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x001D	1000BaseTHD	PDPortDataReal, PDPortDataRealExtended
0x001E	1000BaseTFD	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x001F	10GbaseX	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0020	10GbaseLX4	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0021	10GbaseR	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0022	10GbaseER	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0023	10GbaseLR	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0024	10GbaseSR	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0025	10GbaseW	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0026	10GbaseEW	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0027	10GbaseLW	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0028	10GbaseSW	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0029	10GbaseCX4	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x002A	2BaseTL	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck

Value (hexadecimal)	Meaning	Usage
0x002B	10PassTS	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x002C	100BaseBX10D	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x002D	100BaseBX10U	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x002E	100BaseLX10	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x002F	1000BaseBX10D	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0030	1000BaseBX10U	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0031	1000BaseLX10	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0032	1000BasePX10D	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0033	1000BasePX10U	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0034	1000BasePX20D	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0035	1000BasePX20U	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0036	10GbaseT or 100BasePXFD	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0037	10GbaseLRM	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0038	1000BaseKX	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0039	1000BaseKX4	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x003A	1000BaseKR	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x003B	10G1GbasePRXD1	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x003C	10G1GbasePRXD2	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x003D	10G1GbasePRXD3	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck

Value (hexadecimal)	Meaning	Usage
0x003E	10G1GbasePRXU1	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x003F	10G1GbasePRXU2	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0040	10G1GbasePRXU3	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0041	10GbasePRD1	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0042	10GbasePRD2	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0043	10GbasePRD3	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0044	10GbasePRU1	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0045	10GbasePRU3	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0046	40GbaseKR4	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0047	40GbaseCR4	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0048	40GbaseSR4	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x0049	40GbaseFR	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x004A	40GbaseLR4	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x004B	100GbaseCR10	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x004C	100GbaseSR10	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x004D	100GbaseLR4	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
0x004E	100GbaseER4	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
Useable well known numbers	Speed of 100 Mbit/s (and more) and duplexity full	PDPortDataReal, PDPortDataRealExtended, PDPortDataAdjust, PDPortDataCheck
Unuseable well known numbers	Speed less than 100 Mbit/s or duplexity half	PDPortDataReal, PDPortDataRealExtended
other	Reserved	—

For IEEE 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 639) are two codings for the same MAU type.

Table 639 – MAUType with MAUTypeExtension

Value (hexadecimal)	Meaning	Usage
0x0012 ^a , 0x0100 ^b	100BaseFXFD ^a , POF ^b	PDPorDataRealExtended, PDPorDataAdjust, PDPorDataCheck
^a See Table 638.		
^b See Table 641.		

Table 640 – 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	PDPorDataReal
		10BaseTXHD	
		10BaseTXFD	
		10BaseFLHD	
		10BaseFLFD	
		100BaseTXHD	
		100BaseTXFD (Default)	
		100BaseFXHD	
		100BaseFXFD	
		1000BaseXHD	
		1000BaseXFD	
		1000BaseLXHD	
		1000BaseLXFD	
		1000BaseSXHD	
		1000BaseSXFD	
		1000BaseTHD	
		1000BaseTFD	
		10GbaseFX	
		100BaseLX10	
		100BasePXFD	
Reserved	Down Testing (in some test mode) Unknown (status cannot determined)	Useable well known numbers	Reserved
		Unuseable well known numbers	
Reserved	Reserved		

5.2.13.8 Coding of the field MAUTypeExtension

This field, defined as an administrative number, shall be coded as data type Unsigned16 with the values according Table 641.

This document defines, in addition to the IEEE 802.3, additional MAU types. These MAUTypeExtensions are intended to be derivatives of the MAU types.

Table 641 – MAUTypeExtensions and its corresponding MAUTypes

Value (hexadecimal)	Meaning	Attached MAUType	Comments
0x0000	No SubMAUType	Table 638	The MAUType from Table 638 applies.
0x0001 – 0x00FF	Reserved	—	—
0x0100	POF ^a	100BaseFXFD	Polymeric optical fiber
0x0101 – 0xFFEF	Reserved for SubMAUType	—	Central administrative number to unambiguously distinguish between MAUTypeExtensions.
0xFFFF0 – 0xFFFF	Reserved	—	—

^a See IEC 61158-2.

5.2.13.9 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 642.

Table 642 – 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 643.

Table 643 – 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.

5.2.13.10 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 644.

Table 644 – 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.

5.2.13.11 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 645.

Table 645 – 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

5.2.13.12 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 646.

Table 646 – 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-AnnoucePDU Table 645 and Table 646 shall be combined according to Table 647 to decode the domain boundary.

Table 647 – 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		

5.2.13.13 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 648.

This boundary shall be applied for the first 32 RT_CLASS_2 multicast addresses.

Table 648 – 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

5.2.13.14 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 649.

Table 649 – 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

5.2.13.15 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 650.

Table 650 – 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

5.2.13.16 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 651.

Table 651 – 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.

5.2.13.17 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 652.

Table 652 – 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	PDPortDataReal
0x07	LowerLayerDown	PDPortDataReal
0x08 – 0xFF	Reserved	—

Bit 8 – 15: LinkState.Port

This field shall be set according to Table 653. 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 653 – 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	—

5.2.13.18 Coding of the field MediaType

This field shall be coded as data type Unsigned32 with the values according to Table 654.

Table 654 – MediaType

Value (hexadecimal)	Meaning	Usage
0x00	Unknown	PDPortDataReal
0x01	Copper cable	PDPortDataReal
0x02	Fiber optic cable	PDPortDataReal
0x03	Radio communication	PDPortDataReal
0x04 – 0xFFFFFFFF	Reserved	—

5.2.13.19 Coding of the field MaxBridgeDelay

This field shall be coded as data type Unsigned32 according to Table 655 in nanoseconds. Figure 42 shows the meaning of MaxBridgeDelay.

Table 655 – MaxBridgeDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x0000FFFF	From engineering used bridge delay for RT_CLASS_3 calculation
0x00010000 – 0xFFFFFFFF	Reserved

5.2.13.20 Coding of the field NumberOfPorts

This field shall be coded as data type Unsigned32 with values according to Table 656.

Table 656 – NumberOfPorts

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x000000FF	Number of following port entries
0x00000100 – 0xFFFFFFFF	Reserved

5.2.13.21 Coding of the field MaxPortTxDelay

This field shall be coded as data type Unsigned32 according to Table 657 in nanoseconds. Figure 42 shows the meaning of MaxPortTxDelay.

Table 657 – MaxPortTxDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x0000FFFF	From engineering used port transmit delay for RT_CLASS_3 calculation
0x00010000 – 0xFFFFFFFF	Reserved

5.2.13.22 Coding of the field MaxPortRxDelay

This field shall be coded as data type Unsigned32 according to Table 658 in nanoseconds. Figure 42 shows the meaning of MaxPortRxDelay.

Table 658 – MaxPortRxDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x0000FFFF	From engineering used port receive delay for RT_CLASS_3 calculation
0x00010000 – 0xFFFFFFFF	Reserved

5.2.13.23 Coding of the field MaxLineRxDelay

This field shall be coded as data type Unsigned32 according to Table 659 in nanoseconds. Figure 42 shows the meaning of MaxLineRxDelay.

Table 659 – MaxLineRxDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown No delay, forward as soon as possible
0x00000001 – 0x3B9AC9FF	From engineering used line delay for RT_CLASS_3 calculation
0x3B9ACA00 – 0xFFFFFFFF	Reserved

5.2.13.24 Coding of the field YellowTime

This field shall be coded as data type Unsigned32 according to Table 660 in nanoseconds. Figure 114 and Figure 116 show the meaning and Figure 115 the calculation principle of YellowTime.

The YELLOW period allows a bridge to optimize the forwarding of frames as shown in Figure 115.

If the YellowTime, calculated according Formula (65) and Formula (66), is less than the maximum Ethernet frame, the fragmentation shall be enabled.

NOTE The YellowTime is 125 µs if running without fragmentation.

$$\text{YellowTime} = (\text{InterFrameGAP} + \text{PreambleLength} + \text{StartFrameDelimiter} + \text{MAX}(\text{Selected Protocols})) \times 80 \text{ ns} + \text{REDBeginSafetyMargin} \quad (65)$$

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 length of the frames of the selected protocols
REDBeginSafetyMargin	is the time which covers internal delays of a switch

$$\text{MAX}(\text{Selected Protocols}) = \text{Maximum (PTCP, MRP, RT, IEEE 802.1AS)} \quad (66)$$

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 802.1AS	is the maximum size of a frame used by the IEEE 802.1AS protocol Fragmentable protocol elements are ignored

Table 660 – YellowTime

Value (hexadecimal)	Meaning	Usage
0x00 – 0x1ABF	Reserved	—
0x1AE0	Yellow time in Nanoseconds Equal to 86 Octets at 100 Mbit/s	Fragmentation is supported. In this case Formula (66) 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 <i>not</i> supported. In this case Formula (66) returns the value 1 522 Octets.
0x0x1E849 – 0xFFFFFFFF	Reserved	—

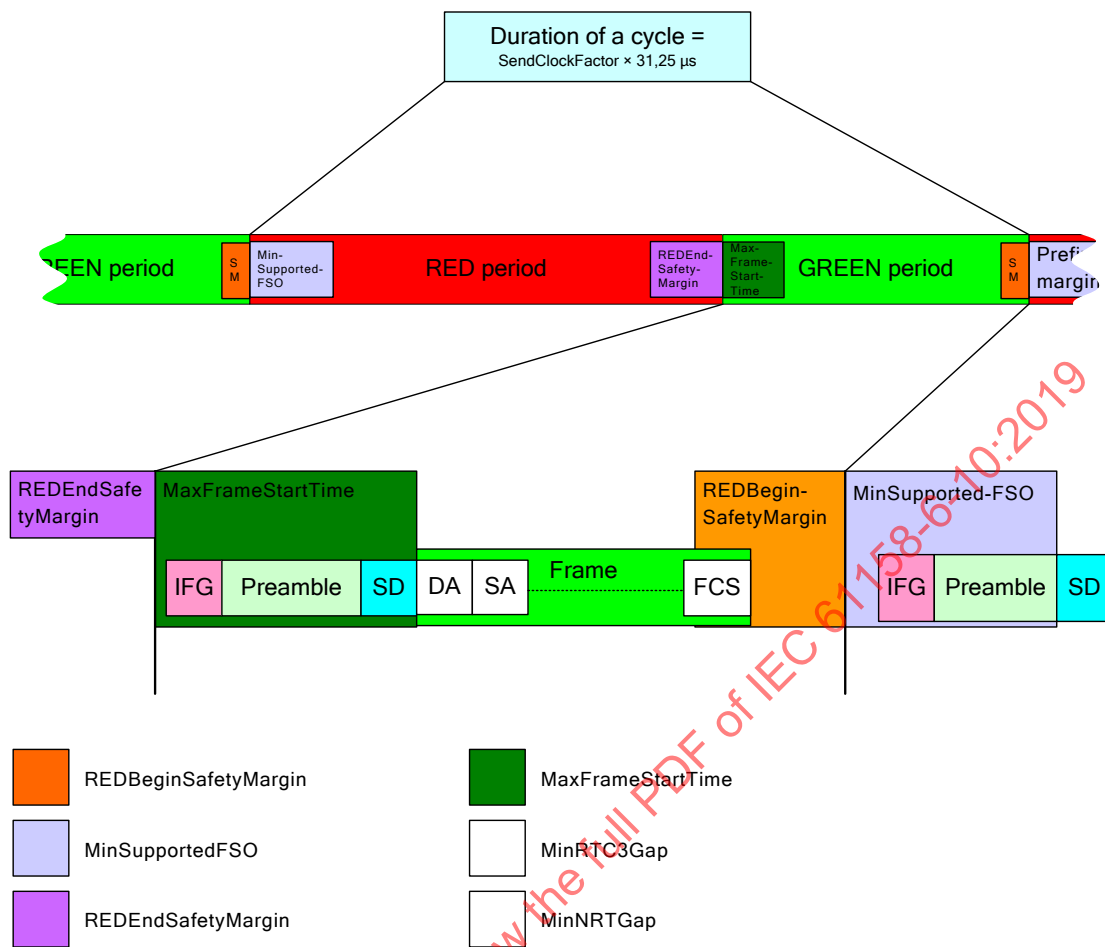


Figure 114 – Calculation principle for a cycle

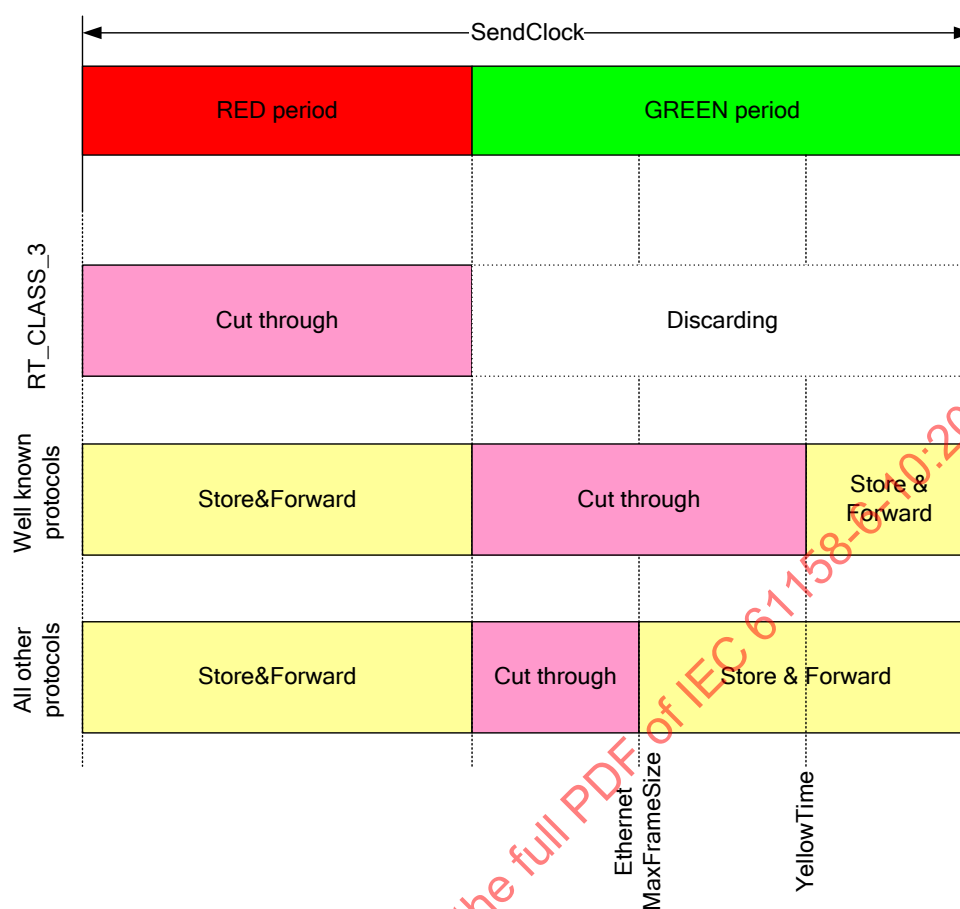


Figure 115 – Calculation principle for the minimum YellowTime

5.2.13.25 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 661 and Table 662.

All RT_CLASS_3 frames, forwarded or local injected shall be inside the range given by StartOfRedFrameID and EndOfRedFrameID.

Table 661 – StartOfRedFrameID in conjunction with ARProperties.StartupMode:= Legacy

Value (hexadecimal)	Meaning
0x0000 – 0x007F	Reserved
0x0080	Reserved Shall be accepted in case of legacy support for PTCIP in RED
0x0081 – 0x00FF	Reserved
0x0100 – 0x0FFF	Mandatory 0x0100 is the default value
0x1000 – 0xFFFF	Reserved

Table 662 – StartOfRedFrameID in conjunction with ARProperties.StartupMode:= Advanced

Value (hexadecimal)	Meaning
0x0000 – 0x00FF	Reserved
0x0100 – 0x0FFF	Mandatory 0x0100 is the default value
0x1000 – 0xFFFF	Reserved

5.2.13.26 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 663 and Table 664.

Table 663 – EndOfRedFrameID

Value (hexadecimal)	Meaning
0x0000 – 0x00FF	Reserved
0x0100 – 0x0FFF	Mandatory 0x0FFF is the default value
0x1000 – 0xFFFF	Reserved

Table 664 – Dependencies of StartOfRedFrameID and EndOfRedFrameID

Value (hexadecimal)	Meaning
StartOfRedFrameID ≤ EndOfRedFrameID	Valid Shall cover the needed FrameID range
StartOfRedFrameID > EndOfRedFrameID	Not used

5.2.13.27 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 665.

Table 665 – 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 – 0x00000010	Optional Number of following assignment entries
0x00000011 – 0xFFFFFFFF	Reserved

If there is no reserved period for a port, than the NumberOfAssignments shall be coded with one and RedOrangePeriodBegin, OrangePeriodBegin and GreenPeriodBegin shall be coded with zero. Furthermore, the NumberOfPhases shall be coded with one and AssignedValueForReservedBegin, AssignedValueForOrangeBegin and AssignedValueForReservedEnd shall be coded with zero.

5.2.13.28 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 666.

Table 666 – 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.

5.2.13.29 Coding of the field PhaseAssignment

The field PhaseAssignment 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 667.

Table 667 – 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 668.

Table 668 – 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 669.

Table 669 – 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.

5.2.13.30 Coding of the field RedOrangePeriodBegin

This field shall be coded as data type Unsigned32 with values according to Table 670 and Table 671. The time base is 1 ns.

Table 670 – Values of RedOrangePeriodBegin

Value (hexadecimal)	Meaning	Use
0	Relative offset to the start of the related cycle	Mandatory
1 – 0x003D08FF	Relative offset to the start of the related cycle	Optional Use not intended
0x003D0900 – 0xFFFFFFFF	Reserved	—

Table 671 – 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	—

5.2.13.31 Coding of the field OrangePeriodBegin

This field shall be coded as data type Unsigned32 with values according to Table 671, Table 672, and Table 673. The time base is 1 ns.

Table 672 – Values of OrangePeriodBegin with ARProperties.StartupMode == Legacy

Value (hexadecimal)	Meaning	Use
0 – 0x003D08FF	Relative offset to the start of the related cycle	Optional
0x003D0900 – 0xFFFFFFFF	Reserved	—

Table 673 – Values of OrangePeriodBegin with ARProperties.StartupMode == Advanced

Value (hexadecimal)	Meaning	Use
GreenPeriodBegin	OrangePeriodBegin shall be equal to GreenPeriodBegin	—
other	Reserved	—

5.2.13.32 Coding of the field GreenPeriodBegin

This field shall be coded as data type Unsigned32 with values according to Table 671 and Table 674. The time base is 1 ns.

Table 674 – Values of GreenPeriodBegin

Value (hexadecimal)	Meaning	Use
0 – 0x0007A120	Relative offset to the start of the related cycle	Mandatory
0x0007A121 – 0x003D08FF	Relative offset to the start of the related cycle	Optional
0x003D0900 – 0xFFFFFFFF	Reserved	—

5.2.13.33 Coding of the field EtherType

This field shall be coded as Unsigned16 as described for the field LT in Table 35. This specification uses the value according to Table 675.

Table 675 – EtherType

Value (hexadecimal)	Meaning
0x8892	This document

5.2.14 Coding section related to Physical Sync Data

5.2.14.1 Coding of the field PTCPLengthSubdomainName

This field shall be coded as data type Unsigned8.

5.2.14.2 Coding of the field PTCPSubdomainName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.15.1.

NOTE The field PTCPSubdomainName is not terminated by zero.

5.2.14.3 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 676.

Table 676 – 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 677.

Table 677 – 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.

5.2.14.4 Coding of the field **ReservedIntervalBegin**

This field shall be coded as data type Unsigned32 with values according to Table 678, Table 679, Table 682, and Figure 116. The time base is one nanosecond.

Table 678 – ReservedIntervalBegin with ARProperties.StartupMode == Legacy

Value (hexadecimal)	Meaning
0x00000000	Begin of ORANGE period
other	Reserved

Table 679 – ReservedIntervalBegin with ARProperties.StartupMode == Advanced

Value (hexadecimal)	Meaning
0x00000000	Mandatory
other	Reserved

5.2.14.5 Coding of the field **ReservedIntervalEnd**

This field shall be coded as data type Unsigned32 with values according to Table 680, Table 681, Table 682, and Figure 116. The time base is one nanosecond.

Table 680 – ReservedIntervalEnd with ARProperties.StartupMode == Legacy

Value (hexadecimal)	Meaning
0x00000001 – 50 % SendClock	End of ORANGE period
other	Reserved

Table 681 – ReservedIntervalEnd with ARProperties.StartupMode == Advanced

Value (hexadecimal)	Meaning
0x00000000	Mandatory
other	Reserved

The dependencies between the field **ReservedIntervalBegin** and the field **ReservedIntervalEnd** are shown in Table 682.

Table 682 – Dependencies of ReservedIntervalBegin and ReservedIntervalEnd

Value (hexadecimal)	Meaning
ReservedIntervalBegin < ReservedIntervalEnd	An ORANGE period is defined
ReservedIntervalBegin = ReservedIntervalEnd = 0	Default
ReservedIntervalBegin > ReservedIntervalEnd	Not used

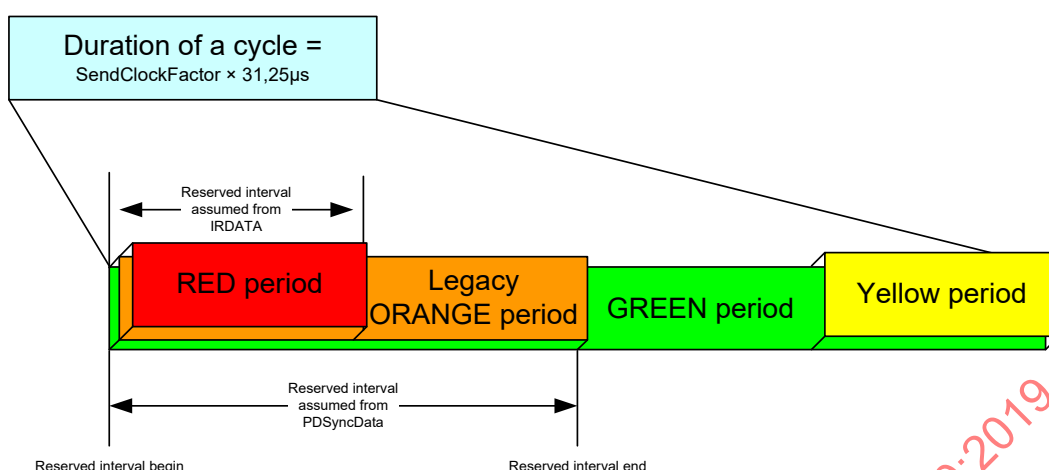


Figure 116 – Definition of the reserved interval

5.2.14.6 Coding of the field SyncSendFactor

This field shall be coded as data type Unsigned32 with values according to Table 683. The time base is 31,25 µs.

Table 683 – 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 (67).

$$\text{SyncSendInterval} = \text{SyncSendFactor} \times 31,25 \mu\text{s} \quad (67)$$

where

SyncSendInterval is the sync send interval

SyncSendFactor is the factor for the calculation of the sync send interval

5.2.14.7 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 684.

Table 684 – 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 (68).

$$\text{Timeout} = \text{PTCPTimeoutFactor} \times \text{SyncSendFactor} \times 31,25 \mu\text{s} \quad (68)$$

where

Timeout	is the time out
PTCPTimeoutFactor	is the PTCP timeout factor for the calculation of the time out
SyncSendFactor	is the factor for the calculation of the sync send interval

5.2.14.8 Coding of the field PTCPTakeoverTimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 685. The time base is the value of the field SyncSendFactor.

Table 685 – 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 (69).

$$\text{Timeout} = \text{PTCPTakeoverTimeoutFactor} \times \text{SyncSendFactor} \times 31,25 \mu\text{s} \quad (69)$$

where

Timeout	is the time out
PTCPTakeoverTimeoutFactor	is the PTCP takeover factor for the calculation of the time out
SyncSendFactor	is the factor for the calculation of the sync send interval
31,25 μs	is the time base for the calculation

5.2.14.9 Coding of the field PTCPMasterStartupTime

This field shall be coded as data type Unsigned16 with the values according to Table 686. The time base is one second.

Table 686 – 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.

5.2.14.10 Coding of the field PLLWindow

This field shall be coded as data type Unsigned32 with the values according to Table 687. The time base shall be one nanosecond.

Table 687 – 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 117 and Figure 118.

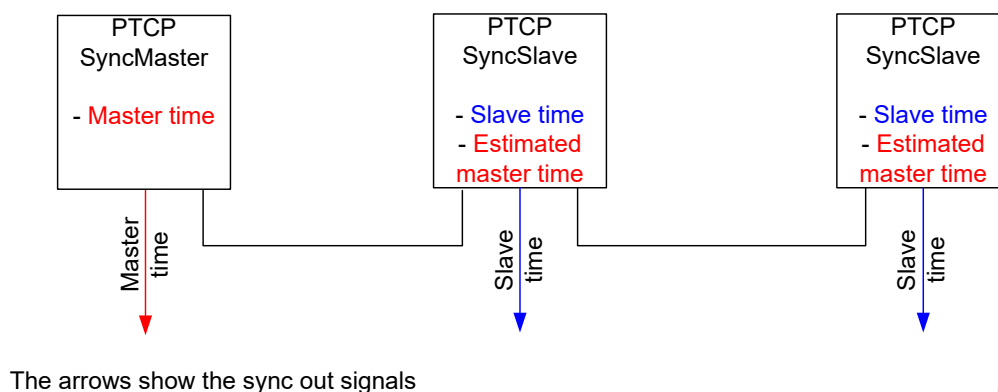


Figure 117 – Toplevel view to the PLL window

NOTE 1 The SyncSlaves estimates the master time by means of PTCP. It controls its slave time to follow the master time, but in real life 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.

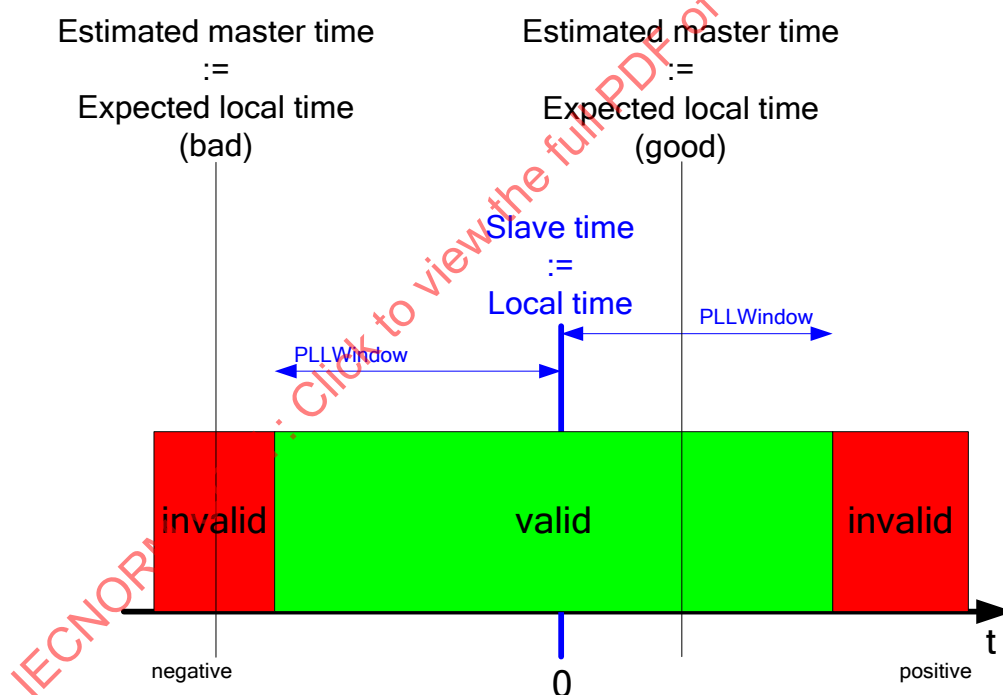


Figure 118 – Definition of PLL window

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 688. The time base is ns.

Table 688 – 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 689. The time base is 31,25 µs.

Table 689 – 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 (70).

$$\text{TimeDC} = \text{TimeDataCycle} \times 31,25 \mu\text{s} \quad (70)$$

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 690. The time base is ns.

Table 690 – 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 691. The time base is ns.

Table 691 – 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 692. The time base is ns.

Table 692 – 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 693. The time base is ns.

Table 693 – 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 694.

Table 694 – ControllerApplicationCycleFactor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x0400	Optional
0x0401 – 0xFFFF	Reserved

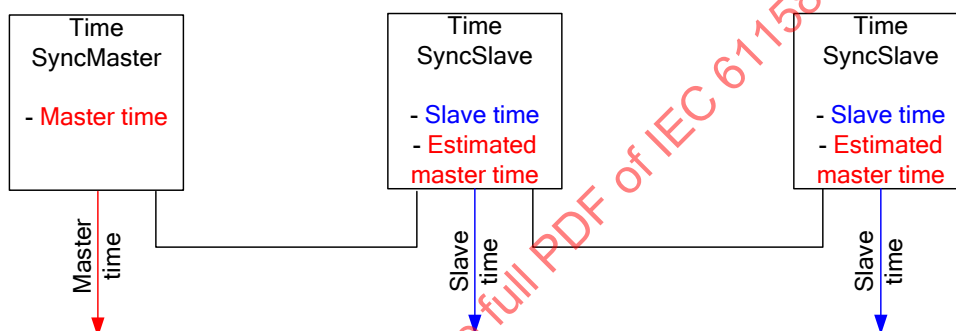
5.2.16 Coding section related to Physical Time Data**5.2.16.1 Coding of the field TimePLLWindow**

This field shall be coded as data type Unsigned32 with the values according to Table 695. The time base shall be one nanosecond.

Table 695 – TimePLLWindow

Value (hexadecimal)	Meaning	SyncID (Time)
0x0000	Disabled	—
0x03E8	Default	Optional (1 µs)
0x2710	Default	Optional (10 µs)
0x000186A0	Default	Mandatory (100 µs)
0x000F4240	Default	Optional (1 ms)
0x00989680	Default	Optional (10 ms)
other	Reserved	—

The definition of the TimePLLWindow is shown in Figure 119 and Figure 120.



The arrows show the sync out signals

Figure 119 – Top level view to the time PLL window

NOTE 1 The SyncSlaves estimates the master time by means of IEEE 802.1AS. It controls its slave time to follow the master time, but in real life 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.

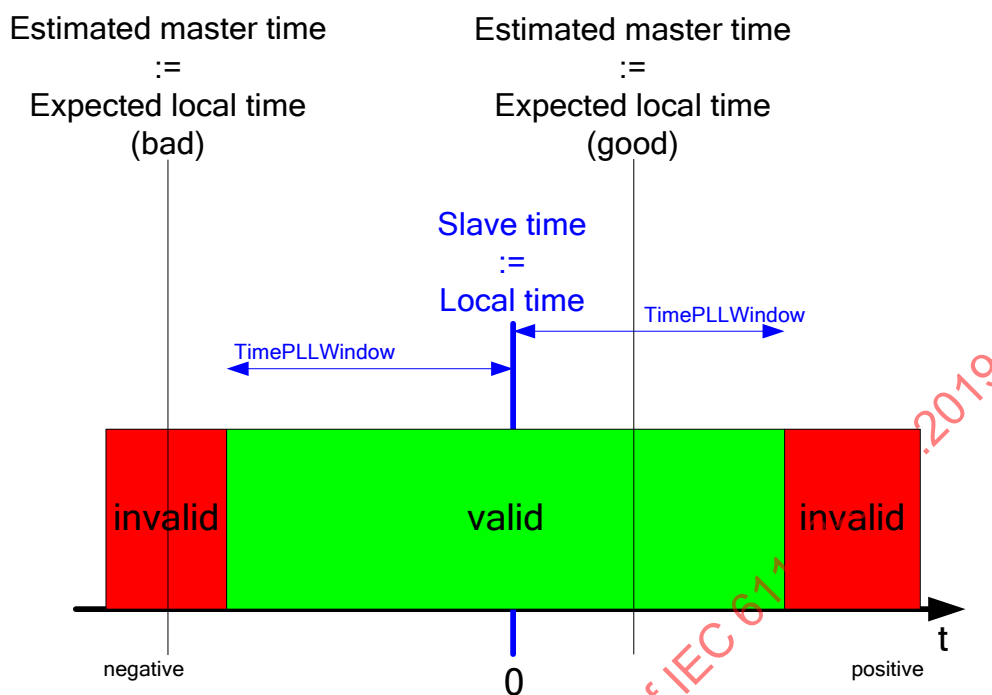


Figure 120 – Definition of time PLL window

5.2.16.2 Coding of the field TimeMasterPriority1

This field shall be coded as Unsigned8 with the value according to Table 696.

Table 696 – 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

5.2.16.3 Coding of the field TimeMasterPriority2

This field shall be coded as Unsigned8 with the value according to Table 697.

Table 697 – TimeMasterPriority2

Value (hexadecimal)	Usage	Meaning
0x00 – 0xF7	Reserved	—
0xF8	Default	Default
0xF9 – 0xFF	Reserved	—

5.2.17 Coding section related to Media Redundancy

5.2.17.1 General

5.2.17.1.1 Void

5.2.17.1.2 Coding of the field MRP_Version

This field shall be coded as data type Unsigned16 and set according to Table 698.

Table 698 – MRP_Version

Value (hexadecimal)	Meaning
0x0001	Version 1
other	Reserved

5.2.17.2 Coding section related to MRP Rings

5.2.17.2.1 Coding of the field MRP_RingState

This field shall be coded as Unsigned16 with the values according to Table 699.

Table 699 – 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	—

5.2.17.2.2 Coding of the field MRP_DomainUUID

This field shall be coded as UUID with the values according to Table 700.

The MRP_DomainUUID can be calculated according to Formula (71).

$$\text{MRP\_DomainUUID} = \text{MD5} (\text{MRP\_DomainName}) \quad (71)$$

where

MRP_DomainUUID	is the MRP domain UUID
MD5	is the IETF RFC 6151 define function to create a 128bit fingerprint
MRP_DomainName	is the variable length input parameter MRP domain name

Table 700 – MRP_DomainUUID

Value (UUID)	Meaning	Usage
00000000-0000-0000-0000-000000000000	—	Reserved
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFFFFE	UUID for media redundancy domain	Optional
FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFFFFF	Default UUID for media redundancy domain	Mandatory

5.2.17.2.3 Coding of the field MRP_LengthDomainName

This field shall be coded as data type Unsigned8 and shall be set according to Table 701.

Table 701 – MRP_LengthDomainName

Value (hexadecimal)	Meaning
0x00	Reserved
0x01 – 0xF0	Allowed values
0xF1 – 0xFF	Reserved

5.2.17.2.4 Coding of the field MRP_DomainName

This field shall be coded as data type OctetString with 1 to 240 octets according to Table 702 and 4.3.1.4.15.2.

Table 702 – MRP_DomainName

Meaning	Usage
Name of the MRP Domain	—

NOTE The field MRP_DomainName is not terminated by zero.

5.2.17.2.5 Coding of the field MRP_Role

This field shall be coded as data type Unsigned16 and shall be set according to Table 703.

Table 703 – 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

5.2.17.2.6 Coding of the field MRP_Version

This field shall be coded as data type Unsigned16 and set according to Table 704.

Table 704 – MRP_Version

Value (hexadecimal)	Meaning
0x0001	Version 1
other	Reserved

5.2.17.2.7 Coding of the field MRP_Prio

This field shall be coded as Unsigned16 and set according to Table 705.

Table 705 – 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

5.2.17.2.8 Coding of the field MRP_TOPchgT

This field shall be coded as data type Unsigned16 and set according to Table 706 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 706 – 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	—

5.2.17.2.9 Coding of the field MRP_TOPNRmax

This field shall be coded as data type Unsigned16 and set according to Table 707.

Table 707 – 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	—

5.2.17.2.10 Coding of the field MRP_TSTshortT

This field shall be coded as data type Unsigned16 and set according to Table 708 with a time base of 1 ms.

Table 708 – 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	—

5.2.17.2.11 Coding of the field MRP_TSTdefaultT

This field shall be coded as data type Unsigned16 and set according to Table 709 with a time base of 1 ms.

Table 709 – 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	—

5.2.17.2.12 Coding of the field MRP_TSTNRmax

This field shall be coded as data type Unsigned16 and set according to Table 710.

Table 710 – 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	—

5.2.17.2.13 Coding of the field MRP_LNKdownT

This field shall be coded as data type Unsigned16 according to Table 711 with a time base of 1 ms.

Table 711 – 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	—

5.2.17.2.14 Coding of the field MRP_LNKupT

This field shall be coded as data type Unsigned16 according to Table 712 with a time base of 1 ms.

Table 712 – 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	—

5.2.17.2.15 Coding of the field MRP_LNKNRmax

This field shall be coded as data type Unsigned16 according to Table 713.

Table 713 – 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	—

5.2.17.2.16 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 714.

Table 714 – MRP_Check.MediaRedundancyManager

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Disable MediaRedundancyManager diagnosis
0x01	ON	Enable MediaRedundancyManager diagnosis (see Table 592)

Bit 1: MRP_Check.MRP_DomainUUID

This field shall be set according to Table 715.

Table 715 – 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 591)

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.

5.2.17.2.17 Coding of the field MRP_NumberOfEntries

This field shall be coded as data type Unsigned8 according to Table 716.

Table 716 – MRP_NumberOfEntries

Value (decimal)	Meaning	Usage
1 – 127	Number of entries	—
other	Reserved	—

5.2.17.2.18 Coding of the field MRP_Instance

This field shall be coded as data type Unsigned8 according to Table 717.

Table 717 – MRP_Instance

Value (decimal)	Meaning	Usage
0 – 126	Internal reference number of a MRP ring parameter set.	—
other	Reserved	—

5.2.17.3 Coding section related to MRP Interconnections

5.2.17.3.1 Coding of the field MRPIC_LengthDomainName

This field shall be coded as data type Unsigned8 according to Table 718.

Table 718 – MRPIC_LengthDomainName

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
0x01 – 0xF0	Length of MRPIC_DomainName in Octets	—
0xF1 – 0xFF	Reserved	—

5.2.17.3.2 Coding of the field MRPIC_DomainName

This field shall be coded as data type UnicodeString8 with a length of one to 240 octets according to Table 719.

Table 719 – MRPIC_DomainName

Meaning	Usage
Manufacturer specific defined name of the MRP Interconnection domain identified by MRPIC_DomainID	—

5.2.17.3.3 Coding of the field MRPIC_State

This field shall be coded as data type Unsigned16 according to Table 720.

Table 720 – MRPIC_State

Value (hexadecimal)	Meaning	Usage
0x00	MRP Interconnection open	—
0x01	MRP Interconnection closed	—
0x0002 – 0xFFFF	Reserved	—

5.2.17.3.4 Coding of the field MRPIC_Role

This field shall be coded as data type Unsigned16 according to Table 721.

Table 721 – MRPIC_Role

Value (hexadecimal)	Meaning	Usage
0x00	No role assigned	—
0x01	MRP Interconnection Client (MIC)	—
0x02	MRP Interconnection Manager (MIM)	—
0x0003 – 0xFFFF	Reserved	—

5.2.17.3.5 Coding of the field MRPIC_DomainID

This field shall be coded as data type Unsigned16 according to Table 722.

Table 722 – MRPIC_DomainID

Value (hexadecimal)	Meaning	Usage
0x0000	No MRP Interconnection Domain defined	—
0x0001 – 0xFFFF	Uniquely administered MRP Interconnection Domain identification	—

5.2.17.3.6 Coding of the field MRPIC_TOPchgT

This field shall be coded as data type Unsigned16 according to Table 723 with a time base of 10 ms.

It contains the interval for sending repetition of interconnection topology change frames for the MIM.

Table 723 – 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	—

5.2.17.3.7 Coding of the field MRPIC_TOPNRmax

This field shall be coded as data type Unsigned16 according to Table 724.

It contains the interval count which controls repeated transmission of interconnection topology change frames for the MIM.

Table 724 – MRPIC_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	—

Formula (72) defines the common point of time to invoke the Flush Filtering Data base service.

$$\text{FlushFDB} = \text{MRPIC\_TOPNRmax} \times \text{MRPIC\_TOPchgT} \quad (72)$$

where

FlushFDB	is 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

5.2.17.3.8 Coding of the field MRPIC_LinkStatusChangeT

This field shall be coded as data type Unsigned16 according to Table 725 with a time base of 1 ms.

It contains the interconnection Link Status poll time interval for the MIM.

Table 725 – 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	—

5.2.17.3.9 Coding of the field MRPIC_LinkStatusNRmax

This field shall be coded as data type Unsigned16 according to Table 726.

It contains the maximum outstanding response to Link Status poll frame count for the MIM. A higher value causes an interconnection failure.

Table 726 – MRPIC_LinkStatusNRmax

Value (decimal)	Meaning	Usage
0 – 7	Reserved	—
8	8 outstanding response to polls	Mandatory (200 ms reconfiguration time)
9 – 100	9 – 100 outstanding response to polls	Optional
101 – 65 535	Reserved	—

5.2.17.3.10 Coding of the field MRPIC_LNKdownT

This field shall be coded as data type Unsigned16 according to Table 727 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 727 – 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	—

5.2.17.3.11 Coding of the field MRPIC_LNKupT

This field shall be coded as data type Unsigned16 according to Table 728 with a time base of 1 ms.

It contains the interval for sending link up Interconnection Link Change frames on ringports and interconnection ports for the MIC.

Table 728 – 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	—

5.2.17.3.12 Coding of the field MRPIC_LNKNRmax

This field shall be coded as data type Unsigned16 according to Table 729.

It contains the Interconnection Link Change frame count which controls repeated transmission of Interconnection Link up or Link down frames for the MIC.

Table 729 – 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	—

5.2.17.3.13 Coding of the field MRPIC_StartDelay

This field shall be coded as data type Unsigned16 according to Table 730 with a time base of 100 ms.

Table 730 – 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 – 0xFFFF	Reserved	—
0xFFFF	Wait infinite time	Optional Startup delay

5.2.17.3.14 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 731.

Table 731 – 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 732.

Table 732 – 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.

5.2.18 Coding section related to fiber optics**5.2.18.1 Coding of the field VendorBlockType**

This field shall be coded as data type Unsigned16 and set according to Table 733.

Table 733 – VendorBlockType

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Vendor specific

5.2.18.2 Coding of the field FiberOpticType

This field shall be coded as data type Unsigned32 and set according to Table 734.

Table 734 – 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.	

5.2.18.3 Coding of the field FiberOpticCableType

This field shall be coded as data type Unsigned32. The coding shall be according to Table 735.

Table 735 – 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 indoor for installation inside or outside of an electric cabinet.	

5.2.18.4 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 736.

Table 736 – 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 737.

Table 737 – FiberOpticPowerBudgetType.CheckEnable

Value (hexadecimal)	Meaning
0x0	OFF
0x1	ON Comparison value is FiberOpticPowerBudgetType.Value

5.2.18.5 Coding of the field FiberOpticManufacturerSpecific

This field shall be coded as data type OctetString.

5.2.19 Coding section related to network components**5.2.19.1 Void****5.2.19.2 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 738.

Table 738 – 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 739.

Table 739 – 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 121 and Figure 122.

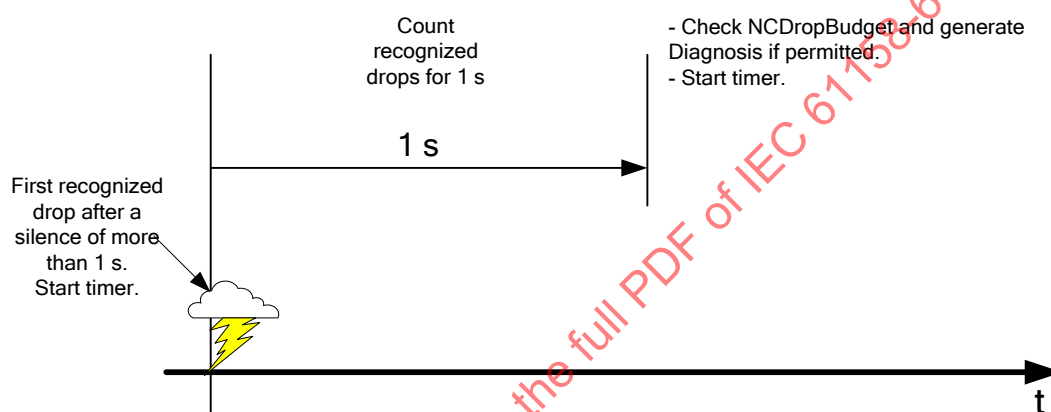


Figure 121 – Detection of dropped frames – appear

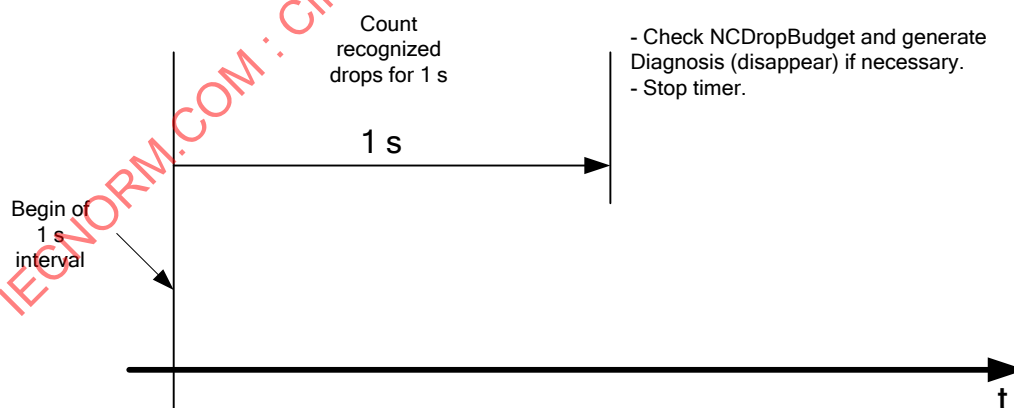


Figure 122 – Detection of dropped frames – disappear

5.2.20 Coding section related port statistic

5.2.20.1 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 740.

Table 740 – 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 741.

Table 741 – CounterStatus.ifOutOctets

Value (hexadecimal)	Meaning
0x00	The content of the field ifOutOctet 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 742.

Table 742 – 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 743.

Table 743 – 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 744.

Table 744 – CounterStatus.ifInErrors

Value (hexadecimal)	Meaning
0x00	The content of the field ifInErrors is valid

Value (hexadecimal)	Meaning
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 745.

Table 745 – 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 746.

Table 746 – CounterStatus.Reserved

Value (hexadecimal)	Meaning
0x00	Reserved

5.2.20.2 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 not supported.

5.2.20.3 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 not supported.

5.2.20.4 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 not supported.

5.2.20.5 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 not supported.

5.2.20.6 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 not supported.

5.2.20.7 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 not supported.

5.2.21 Coding section related to fast startup

5.2.21.1 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 747.

Table 747 – 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.

5.2.21.2 Coding of the field FSHelloInterval

This field shall be coded as data type Unsigned32. The coding shall be according to Table 748.

Table 748 – FSHelloInterval

Value (hexadecimal)	Meaning	Usage
0x00000001E	30 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	Default
0x000000032	50 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
0x000000064	100 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
0x00000012C	300 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
0x0000001F4	500 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
0x0000003E8	1 000 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	—
other	Reserved	—

5.2.21.3 Coding of the field FSHelloRetry

This field shall be coded as data type Unsigned32. The coding shall be according to Table 749.

Table 749 – 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	—

5.2.21.4 Coding of the field FSHelloDelay

This field shall be coded as data type Unsigned32. The coding shall be according to Table 750.

Table 750 – FSHelloDelay

Value (hexadecimal)	Meaning	Usage
0x00000000	OFF	Default
0x00000032	50 ms Wait this time after the first LinkUp.ind before conveying a DCP_Hello.req	—
0x00000064	100 ms Wait this time after the first LinkUp.ind before conveying a DCP_Hello.req	—
0x000001F4	500 ms Wait this time after the first LinkUp.ind before conveying a DCP_Hello.req	—
0x000003E8	1 000 ms Wait this time after the first LinkUp.ind before conveying a DCP_Hello.req	—
other	Reserved	—

5.2.21.5 Coding of the field FSPParameterMode

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: FSPParameterMode.Mode

This field shall be set according to Table 751.

Table 751 – FSPParameterMode.Mode

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	ON	—
0x02	Reserved	—
0x03	Reserved	—

Bit 2 – 23: FSPParameterMode.Reserved_1

This field shall be set according to 3.4.2.2.

Bit 24 – 31: FSPParameterMode.Reserved_2

This field shall be set to zero.

5.2.21.6 Coding of the field FSPParameterUUID

This field shall be coded as data type UUID with values according to Table 752.

Table 752 – FSPParameterUUID

Value (UUID)	Meaning
00000000-0000-0000-0000-000000000000	Reserved
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFFFF	UUID for the record data (including physical device data, ...) delivered between IODConnectRes and IODControlReq.

5.2.22 Coding section related to DFP**5.2.22.1 Coding of the field NumberOfSubframeBlocks**

This field shall be coded as data type Unsigned16 with values according to Table 753.

Table 753 – NumberOfSubframeBlocks

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x00FF	Number of the following SubframeBlocks
0x0100 – 0xFFFF	Reserved

5.2.22.2 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 754.

Table 754 – SFIOCRProperties.DistributedWatchDogFactor

Value (hexadecimal)	Meaning	Description
0x00 – 0x02	Reserved	—
0x03 – 0x1F	Mandatory	An expiration should be signaled to the application as a hint for a possible communication disturbance.
0x20 – 0xFF	Reserved	—

The time base is the SendClockFactor multiplied with ReductionRatio of the monitored provider.

The DistributedWatchDogTime shall be calculated according to Formula (73).

$$\text{Distributed-WatchDogTime} = \frac{\text{Distributed-WatchDogFactor}}{\text{ReductionRatio}} \times \text{SendClockFactor} \times 31,25 \mu\text{s} \quad (73)$$

where

DistributedWatchDogTime is the distributed watch dog time
DistributedWatchDogFactor is the factor for the calculation of the distributed watch dog
SendClockFactor is the send clock factor
ReductionRatio is the reduction ratio

Bit 8 – 15: SFIOCRProperties.RestartFactorForDistributedWD

This field shall be set according to Table 755.

Table 755 – 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 is 100 ms. The RestartForDistributedWDTime shall be calculated according to Formula (74).

$$\text{RestartForDistributed-WDTime} = \text{RestartFactorFor-DistributedWDFactor} \times 100 \text{ ms} \quad (74)$$

where

RestartForDistributedWDTime is the restart for distributed watch dog time
RestartFactorForDistributedWDFactor is the factor for the calculation of the restart for distributed watch dog 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 756.

Table 756 – 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 757.

Table 757 – 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 758.

Table 758 – 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 one, 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 one, shall contain the inverse ordering of the SubframeData.

Bit 31: SFIOCRProperties.SFCRC16

This field shall be set according to Table 759.

Table 759 – 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.

5.2.22.3 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

Bit 0 – 6: SubframeData.Position

This field shall be set according to Table 760.

Table 760 – 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 761.

Table 761 – 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.

5.2.22.4 Frame late error

The checking of the DFP frame late error shall be done according to 4.12.8.4.5.4. Figure 123 and Table 762 show the behavior as an overview.

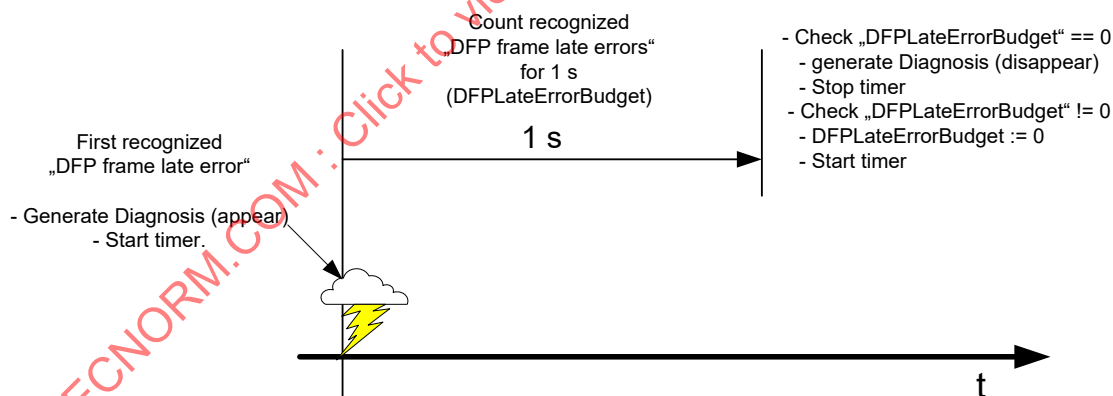


Figure 123 – Detection of DFP late error – appear and disappear

Table 762 – 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 (60)) 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 (60)) 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 show the principle. The rules for ChannelProperties.Specifier apply.	

5.2.22.5 Coding of the field SubframeOffset

This field shall be coded as data type Unsigned16 with values according to Table 763.

Table 763 – 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

5.2.23 Coding section related to MRPD

5.2.23.1 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. It shall be activated if MRPD is used.

5.2.23.2 MediaRedundancyWatchDog expired

The checking whether the MediaRedundancyWatchDog (as an example with the time value 1 s) is expired shall be done according to Figure 124 and Table 764.

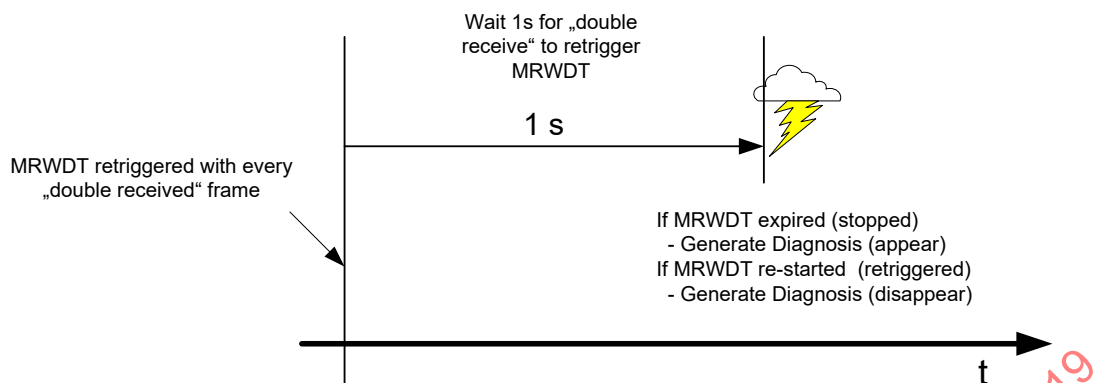


Figure 124 – MediaRedundancyWatchDog expired – appear and disappear

Table 764 – 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 (61)) 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 (61)) 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 show the principle. The rules for ChannelProperties.Specifier apply.	

5.2.24 Coding section related to auto configuration

5.2.24.1 Coding of the field SendClockFactorArray

The field SendClockFactorArray is an array of 16 SCFEntry octets. It shall be used to code the supported SendClockFactors. Each bit of the array corresponds with one SendClockFactor and shall be calculated as shown in Formula (75) and Formula (76).

$$N = (\text{SendClockFactor} - 1) \text{ DIV } 8 \quad (75)$$

where

N	is the number of the octet
SendClockFactor	is the supported SendClockFactor
8	is the bit size of a octet

$$M = (\text{SendClockFactor} - 1) \text{ MOD } 8 \quad (76)$$

where

M	is the number of the bit within the octet
SendClockFactor	is the supported SendClockFactor
8	is the bit size of a octet

NOTE The term DIV stands for division without remainder. The term MOD stands for the remainder of the division.

The SendClockFactorArray shall be structured in ascending order without gaps.

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

Bit 0: SCFEntry_0

This bit shall be set with the values according to Table 765 if the SendClockFactor with the number that meets Formula (75) and Formula (76) is present.

Table 765 – SCFEntry

Value (hexadecimal)	Meaning
0x00	SendClockFactor not supported
0x01	SendClockFactor supported

Bit 1: SCFEntry_1

This bit shall be set with the values according to Table 765 if the SendClockFactor with the number that meets Formula (75) and Formula (76) is present.

Bit 2: SCFEntry_2

This bit shall be set with the values according to Table 765 if the SendClockFactor with the number that meets Formula (75) and Formula (76) is present.

Bit 3: SCFEntry_3

This bit shall be set with the values according to Table 765 if the SendClockFactor with the number that meets Formula (75) and Formula (76) is present.

Bit 4: SCFEntry_4

This bit shall be set with the values according to Table 765 if the SendClockFactor with the number that meets Formula (75) and Formula (76) is present.

Bit 5: SCFEntry_5

This bit shall be set with the values according to Table 765 if the SendClockFactor with the number that meets Formula (75) and Formula (76) is present.

Bit 6: SCFEntry_6

This bit shall be set with the values according to Table 765 if the SendClockFactor with the number that meets Formula (75) and Formula (76) is present.

Bit 7: SCFEntry_7

This bit shall be set with the values according to Table 765 if the SendClockFactor with the number that meets Formula (75) and Formula (76) is present.

5.2.24.2 Coding of the field ACCommunicationProperties

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: ACCommunicationProperties.DFP

This field shall be set according to Table 766.

Table 766 – ACCommunicationProperties.DFP

Value (hexadecimal)	Meaning
0x00	Not supported
0x01	Supported

Bit 1 – 2: ACCommunicationProperties.RTC3

This field shall be set according to Table 767.

Table 767 – ACCommunicationProperties.RTC3

Value (hexadecimal)	Meaning
0x00	Not supported
0x01	PDIRFrameData with local and forwarding (including the FrameSendOffset) information needed
0x02	PDIRFrameData with local information needed
0x03	Reserved

Bit 3: ACCommunicationProperties.RTCUDP

This field shall be set according to Table 768.

Table 768 – ACCommunicationProperties.RTCUDP

Value (hexadecimal)	Meaning
0x00	RT_CLASS_UDP not supported
0x01	RT_CLASS_UDP supported

Bit 3 – 15: ACCommunicationProperties.Reserved_1

This field shall be set according to 3.4.2.2.

Bit 16 – 31: ACCommunicationProperties.Reserved_2

This field shall be set to zero.

5.2.24.3 Coding of the field ACMinDeviceInterval

This field shall be coded as data type Unsigned16 according to Table 769.

Table 769 – ACMinDeviceInterval

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	31,25 µs MinDeviceInterval as a factor of 31,25 µs.
...	...
0x1000	128 ms MinDeviceInterval as a factor of 31,25 µs.
0x1001 – 0xFFFF	Reserved

5.2.25 Coding section related to controller to controller communication**5.2.25.1 Coding of the field FromOffsetData**

This field shall be coded as data type Unsigned32 according to Table 770.

Table 770 – 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.

5.2.25.2 Coding of the field NextOffsetData

This field shall be coded as data type Unsigned32 according to Table 771.

Table 771 – NextOffsetData

Value (hexadecimal)	Meaning	Description
0x00000000	Mandatory	The last chunk of data is read.
Other	Mandatory	Delivers the next to be used offset value for convenience. More data is available.

5.2.25.3 Coding of the field TotalSize

This field shall be coded as data type Unsigned32 according to Table 772.

Table 772 – TotalSize

Value (hexadecimal)	Meaning	Description
0x00000000	Reserved	—
Other	Mandatory	Contains the total size in octets of the requested object.

5.2.26 Coding section related to system redundancy

5.2.26.1 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 773, Figure 125, and Figure 126.

Table 773 – RedundancyInfo.EndPoint1

Value (hexadecimal)	Meaning
0x00	The mounting position of the delivering node in a rack is not the left or above one compared to EndPoint2.
0x01	The mounting position of the delivering node in a rack is the left or above one compared to EndPoint2.

Bit 1: RedundancyInfo.EndPoint2

This field shall be coded according to Table 774, Figure 125, and Figure 126.

Table 774 – RedundancyInfo.EndPoint2

Value (hexadecimal)	Meaning
0x00	The mounting position of the delivering node in a rack is not the right or below one compared to EndPoint1.
0x01	The mounting position of the delivering node in a rack is the right or below one compared to EndPoint1.

Valid combinations of RedundancyInfo.EndPoint1 and RedundancyInfo.EndPoint2 are shown in Table 775.

Table 775 – 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 125 and Figure 126 show the naming scheme for EndPoint1 and Endpoint2.

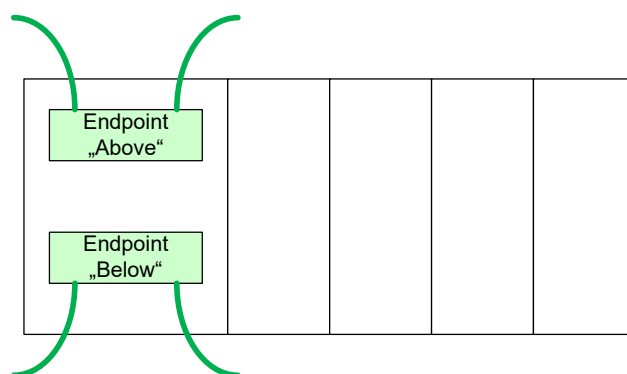


Figure 125 – EndPoint1 and Endpoint2 scheme – above and below

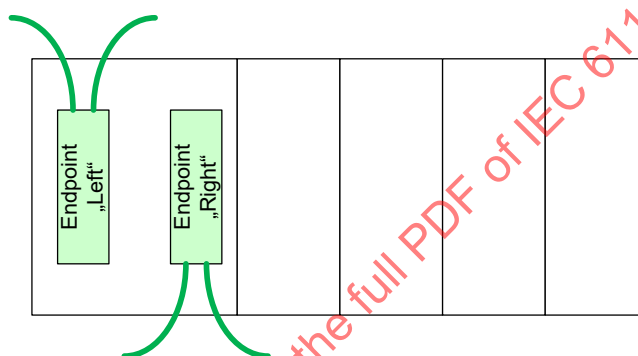


Figure 126 – EndPoint1 and Endpoint2 scheme – left and right

Bit 2 – 15: RedundancyInfo.Reserved

This field shall be set to zero.

5.2.26.2 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 776 and Table 777.

Table 776 – 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 777 – 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 778.

Table 778 – 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 779.

Table 779 – SRProperties.Mode

Value (hexadecimal)	Meaning
0x00	The IO controller do not support APDU_Status.DataStatus.DataValid == Invalid if input data is request as not valid.
0x01	Default The IO device shall use APDU_Status.DataStatus.DataValid == Invalid if input data is request 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.

5.2.26.3 Coding of the field RedundancyDataHoldFactor

This field shall be coded as data type Unsigned16 according to Table 780. The time base shall be 1 ms.

Table 780 – 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 (77).

$$\text{RDHT} = \text{RedundancyDataHoldFactor} \times 1 \text{ ms} \quad (77)$$

where

RDHT is the redundancy data hold time

RedundancyDataHoldFactor is the factor for the calculation of the redundancy data hold time

5.2.26.4 Coding of the field NumberOfEntries

This field shall be coded as data type Unsigned16 according to Table 781.

Table 781 – NumberOfEntries

Value (hexadecimal)	Meaning	Use
0x0000 – 0xFFFF	Number of entries	—

5.2.27 Coding section related to energy saving

5.2.27.1 Void

5.2.27.2 Coding of the field PE_OperationalMode

This field shall be coded as data type Unsigned8 and shall contain the values according to Table 782.

Table 782 – 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	—

5.2.28 Coding section related to asset management

5.2.28.1 Coding section related to AM_Location

5.2.28.1.1 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 783.

Table 783 – 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 784.

The coding “Level not used” shall only be used, if all subsequent levels are not used either.

Table 784 – AM_Location.Levelx

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 784.

Bit 28 – 37: AM_Location.Level2

This field shall be coded according to Table 784.

Bit 38 – 47: AM_Location.Level3

This field shall be coded according to Table 784.

Bit 48 – 57: AM_Location.Level4

This field shall be coded according to Table 784.

Bit 58 – 67: AM_Location.Level5

This field shall be coded according to Table 784.

Bit 68 – 77: AM_Location.Level6

This field shall be coded according to Table 784.

Bit 78 – 87: AM_Location.Level7

This field shall be coded according to Table 784.

Bit 88 – 97: AM_Location.Level8

This field shall be coded according to Table 784.

Bit 98 – 107: AM_Location.Level9

This field shall be coded according to Table 784.

Bit 108 – 117: AM_Location.Level10

This field shall be coded according to Table 784.

Bit 118 – 127: AM_Location.Level11

This field shall be coded according to Table 784.

5.2.28.1.2 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 783.

Bit 8 – 15: AM_Location.Reserved1

This field shall be coded according to Table 785.

Table 785 – 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 786 with the SubslotNumber at which the reported asset starts.

Table 786 – AM_Location.BeginSubslotNumber

Value (hexadecimal)	Meaning
0x0000 – 0xFFFFE	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 787 with the SubslotNumber at which the reported asset ends.

Table 787 – AM_Location.EndSubslotNumber

Value (hexadecimal)	Meaning
0x0000 – 0xFFFFE	See 5.2.3.3
0xFFFF	Asset covers whole slot

Bit 80 – 95: AM_Location.Reserved2

This field shall be coded according to Table 788.

Table 788 – AM_Location.Reserved2

Value (hexadecimal)	Meaning
0x0000	Reserved

Bit 96 – 111: AM_Location.Reserved3

This field shall be coded according to Table 789.

Table 789 – AM_Location.Reserved3

Value (hexadecimal)	Meaning
0x0000	Reserved

Bit 112 – 127: AM_Location.Reserved4

This field shall be coded according to Table 790.

Table 790 – AM_Location.Reserved4

Value (hexadecimal)	Meaning
0x0000	Reserved

5.2.28.2 Coding section related to asset management string**5.2.28.2.1 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 “ ” (blank).

5.2.28.2.2 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.

5.2.28.2.3 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.

5.2.28.2.4 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 791 and Table 792. The definition is dependent on the AM_DeviceIdentification.DeviceID.

Table 791 – AM_DeviceIdentification.DeviceSubID

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Usage defined by AM_DeviceIdentification.Organization

**Table 792 – 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 793. The definition is dependent on the AM_DeviceIdentification.VendorID.

Table 793 – 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 794. The definition is dependent on the AM_DeviceIdentification.Organization.

Table 794 – 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 795.

Table 795 – 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

5.2.28.3 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 562.

Any manufacturer specific values shall be defined according to AM_DeviceIdentification.

5.2.29 Coding section related to reporting system

5.2.29.1 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.

5.2.29.2 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 796.

Table 796 – 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.

5.2.29.3 Coding of the field RS_BlockType

This field shall be coded as data type Unsigned16 with the values according to Table 797 and Table 798.

Table 797 – RS_BlockType used for events

Value (hexadecimal)	Name	Comment
0x0000	Reserved	—
0x0001 – 0x3FFF	Manufacturer specific	—

Value (hexadecimal)	Name	Comment
0x4000	RSO_StatusObserver	Observer status observer Checks whether a given observer is running normally. If not, an event is posted.
0x4001	RSO_BufferObserver	Buffer status observer Checks whether the RS event buffer is running normally. If for example an overflow occurs, an event is posted.
0x4002	RSO_TimeStatus	Time status observer Checks whether the source of time is running normally. If for example synchronization is lost, an event is posted.
0x4003	RSO_SRLObserver	System redundancy layer observer Checks whether a switchover occurs. When switching over events are posted showing begin and end of the switch over.
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 inputs changes and for example posts an event for each change.
0x4011 – 0x6FFF	Reserved for normative usage	—
0x7000 – 0x7FFF	Reserved for profile usage	—

Table 798 – 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

5.2.29.4 Coding of the field RS_BlockLength

This field shall be coded as data type Unsigned16 with the values according to Table 799 and Table 800.

Table 799 – 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 800 – 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

5.2.29.5 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 801.

By means of the RS_Specifier.SequenceNumber the sink detects duplications and the source detects which events are acknowledged. It should start with zero and shall be incremented by one 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 801 – 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 802.

Table 802 – RS_Specifier.Specifier

Value (hexadecimal)	Meaning
0x0	Current value
0x1	Appears
0x2	Disappears
0x3	Reserved

5.2.29.6 Coding of the field RS_MinusError

This field shall be coded as data type Unsigned16 with the values according to Table 803. It contains the maximum negative event capture error defined by the device implementation in relation to the local maintained time (TIME ASE).

Table 803 – RS_MinusError

Value (hexadecimal)	Comment
0x0000 – 0x2710	Maximum negative deviation of the Time_TimeStamp in units of 10 µs
0x2711 – 0xFFFF	Reserved

5.2.29.7 Coding of the field RS_PlusError

This field shall be coded as data type Unsigned16 with the values according to Table 804. It contains the maximum positive event capture error defined by the device implementation in relation to the local maintained time (TIME ASE).

Table 804 – RS_PlusError

Value (hexadecimal)	Comment
0x0000 – 0x2710	Maximum positive deviation of the Time_TimeStamp in units of 10 µs
0x2711 – 0xFFFF	Reserved

5.2.29.8 Coding of the field RS_ExtensionBlockType

This field shall be coded as data type Unsigned8 with the values according to Table 805. Each RS_BlockType creates its own RS_ExtensionBlockType namespace.

Table 805 – 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

5.2.29.9 Coding of the field RS_ExtensionBlockLength

This field shall be coded as data type Unsigned8 with the values according to Table 806.

Table 806 – RS_ExtensionBlockLength

Value (hexadecimal)	Comment
0x00 – 0x01	Reserved
0x02 – 0x2F	Length of Data
0x30 – 0xFF	Reserved

5.2.29.10 Coding of the field RS_MaxScanDelay

This field shall be coded as data type Unsigned16 with the values according to Table 807.

Table 807 – RS_MaxScanDelay

Value (hexadecimal)	Comment
0x0000	Reserved
0x0001 – 0x2710	Maximum scan delay for the supervised channel in 10 µs.
0x2711 – 0xFFFF	Reserved

5.2.29.11 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 808.

Table 808 – RS_AdjustSpecifier.Incident

Value (hexadecimal)	Comment
0x00	Reserved
0x01	Rising edge
0x02	Falling edge
0x03	Reserved

5.2.29.12 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 809.

Table 809 – 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 810.

Table 810 – RS_ReasonCode.Detail

Value (hexadecimal)	Comment
0x0000	No detail
0x0001 – 0xFFFF	Reserved

5.2.29.13 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 811.

Table 811 – 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.

5.2.29.14 Coding of the field RS_DomainIdentification

This field shall be coded as data type OctetString[16] with the values according to Table 812.

Table 812 – RS_DomainIdentification

Value (hexadecimal)	Comment
zero	Unknown the reporting system uses the local (arbitrary timescale) time or no domain known
other	Domain identification, i.e. a Domain UUID

5.2.29.15 Coding of the field RS_MasterIdentification

This field shall be coded as data type OctetString[8] with the values according to Table 813.

Table 813 – RS_MasterIdentification

Value (hexadecimal)	Comment
zero	Unknown, the reporting system uses the local (arbitrary timescale) time
other	Master identification, i.e. the MAC-address of the synchronization master in EUI-64 format

The RS_DomainIdentification and RS_MasterIdentification show the state when the event was detected.

5.2.30 Coding section related to Logbook

5.2.30.1 Coding of the field ActualLocalTimeStamp

This field shall be coded as data type Unsigned64 according to Table 814.

Table 814 – ActualLocalTimeStamp

Value (hexadecimal)	Meaning	Use
0x0000000000000000 – 0xFFFFFFFFFFFFFFFF	Contains the current cycle count value when reading the log book.	—

5.2.30.2 Coding of the field LocalTimeStamp

This field shall be coded as data type Unsigned64 according to Table 815.

Table 815 – LocalTimeStamp

Value (hexadecimal)	Meaning	Use
0x0000000000000000 – 0xFFFFFFFFFFFFFFFF	Contains the current cycle count when storing the entry to the log book.	—

5.2.30.3 Coding of the field NumberOfLogEntries

This field shall be coded as data type Unsigned16 according to Table 816.

Table 816 – NumberOfLogEntries

Value	Meaning	Use
0	Reserved	—
other	Number of log entries	—

5.2.30.4 Coding of the field EntryDetail

This field shall be coded as data type Unsigned32 according to protocol machine behavior and Table 817.

Table 817 – EntryDetail

Value (hexadecimal)	Meaning	Use
0x00000000	No detail	—
other	Value derived from the signaling protocol machine	—

5.2.31 Coding section related to Time

5.2.31.1 Void

5.2.31.2 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 818.

Table 818 – Time_TimeStamp

Comment
Point in time fetched from TIME ASE when the event occurs.

5.2.32 Coding section related to Channel Related Process Alarm Reason

5.2.32.1 Overview

Allowed combinations of the fields defined for the Channel Related Process Alarm Reason functionality are shown in Table 819.

Table 819 – 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

5.2.32.2 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 820.

Table 820 – PRAL_ChannelProperties.Reserved_1

Value (hexadecimal)	Meaning	Usage
0x00 – 0xFF	Reserved	—

Bit 8: PRAL_ChannelProperties.Accumulative

This field shall be set according to Table 821.

Table 821 – PRAL_ChannelProperties.Accumulative

Value (hexadecimal)	Meaning	Usage
0x00	Single	Single channel Reason only for the reported channel
0x01	Accumulative	Multiple channel Accumulative reason from more than one channel

Bit 9 – 12: PRAL_ChannelProperties.Reserved_2

This field shall be set according to Table 822.

Table 822 – 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 823.

Table 823 – PRAL_ChannelProperties.Direction

Value (hexadecimal)	Meaning
0x00	Manufacturer specific
0x01	Input
0x02	Output
0x03	Input/Output
0x04 – 0x07	Reserved

5.2.32.3 Coding of the field PRAL_Reason

This field shall be coded as data type Unsigned16 with the values according to Table 824.

Table 824 – 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)
0x0005	Compare event for DQ0 has occurred	Compare of counter or way measurement register with a parameterized value "DQ0" shows hit (Compare event for DQ0 has occurred)

Value (hexadecimal)	Meaning	Assigned text
0x0006	Compare event for DQ1 has occurred	Compare of counter or way measurement register with a parameterized value "DQ1" shows hit (Compare event for DQ1 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 common profiles ^a	Profile specific reason
0xA000 – 0xFFFF	Reserved	Reserved
^a Central administrative number to unambiguously distinguish between profiles		

5.2.32.4 Coding of the field PRAL_ExtReason

This field shall be coded as data type Unsigned16 with the values according to Table 825. The value of this field depends on the field PRAL_Reason.

Table 825 – 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

5.2.32.5 Coding of the field PRAL_ReasonAddValue

This field shall be coded as data type OctetString[] with 4 to 128 octets according to Table 826, Table 827, Table 828, and Table 829.

Table 826 – Usage of PRAL_ReasonAddValue

Length (hexadecimal)	Meaning
0x04	Definition from Table 827 applies
0x05 – 0x80	Definition from Table 828 applies

Table 827 – Values for PRAL_ReasonAddValue[0..3]

Value (hexadecimal)	Meaning
0x00, 0x00, 0x00, 0x00	No additional information – independent from PRAL_Reason and PRAL_ExtReason
other	Definition depending PRAL_Reason and PRAL_ExtReason

Table 828 – Values for PRAL_ReasonAddValue[0] to [127]

Value (hexadecimal)	Meaning
0x00 – 0xFF	Definition depending PRAL_Reason and PRAL_ExtReason

5.2.33 PDU checking rules

5.2.33.1 General

5.2.33.1.1 Overview

The following rules shall be applied to check FAL PDUs at the receiver. PDU code checking rules provide redundant information in a condensed way. In case of a contradiction of PDU code checking rules with other parts of the specification, other parts of the specification shall have precedence.

NOTE A PDU is structured with blocks which are identified by their BlockType. Unknown or unexpected blocks lead to a structure fault.

5.2.33.1.2 Additional checking rules, given by profiles

Application profiles as “process automation” or “energy saving” define additional PDU code checking rules. Thus, a parameter shown as valid in the following condensed way of the PDU code checking rules may be signaled as invalid in their context.

5.2.33.2 IODConnectReq

5.2.33.2.1 ArgsLength

This field shall be checked according to Table 829.

Table 829 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	Okay, check IODConnectReq-PDU
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	—
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataRequest	ArgsLength	!= Sum of all BlockLength + Number of Blocks × 4	CMRPC: ArgsLength invalid
IODConnectReq	ARBlockReq	Not first block	RSP-, 0xDB, 0x81, 0x01, 0
IODConnectReq		Case ARProperties.DeviceAccess == 1: other blocks than ARBlockReq	CMRPC: Unknown blocks
IODConnectReq	IOCRBlockReq	Case ARProperties.Device- Access == 0: Number of IOCRBlockReq with CRType == “Input CR” == 0 OR Number of IOCRBlockReq with CRType == “Output CR” == 0	CMRPC: IOCR missing
IODConnectReq	AlarmCRBlockReq	Case ARProperties.Device- Access == 0: Number of AlarmCRBlockReq != 1	CMRPC: Wrong AlarmCRBlock count

5.2.33.2.2 ARBlockReq

This field shall be checked according to Table 830.

Table 830 – ARBlockReq – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0101	RSP-, 0xDB, 0x81, 0x01, 0
BlockLength	!= 54 + StationNameLength	RSP-, 0xDB, 0x81, 0x01, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x01, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x01, 3
ARType	== reserved OR not supported	RSP-, 0xDB, 0x81, 0x01, 4
ARUUID	== NIL	RSP-, 0xDB, 0x81, 0x01, 5
CMInitiatorMacAdd	If (IOCRProperties.RTClass == RT_CLASS_UDP) AND (CMInitiatorMacAdd != zero)	RSP-, 0xDB, 0x81, 0x01, 7
CMInitiatorMacAdd	== multicast address	RSP-, 0xDB, 0x81, 0x01, 7
CMInitiatorObjectUUID	<see next 5 lines>	
.. time low	!= 0xDEA00000	RSP-, 0xDB, 0x81, 0x01, 8
.. time mid	!= 0x6C97	RSP-, 0xDB, 0x81, 0x01, 8
.. time high version	!= 0x11D1	RSP-, 0xDB, 0x81, 0x01, 8
.. Clock[2]	!= 0x82, 0x71	RSP-, 0xDB, 0x81, 0x01, 8
.. Node[6] (contains instance, device, vendor)	Not checked	—
ARProperties.State	ARProperties.State != Active	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.Supervisor- TakeoverAllowed	Don't care	—
ARProperties.Parameterization Server	== External PrmServer	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.Reserved_1	Not checked	—
ARProperties.DeviceAccess	== 1 AND ARType != IOSAR	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.CompanionAR	== 3	RSP-, 0xDB, 0x81, 0x01, 9
CMInitiatorActivityTimeoutFact or	!= 1 to 1 000 (base 100 ms)	RSP-, 0xDB, 0x81, 0x01, 10
InitiatorUDPRTPort	If (IOCRProperties.RTClass == RT_CLASS_UDP) == 0 to 0x03FF else ignore	RSP-, 0xDB, 0x81, 0x01, 11
StationNameLength	== 0 OR > 240 AND fits with BlockLength	RSP-, 0xDB, 0x81, 0x01, 12
CMInitiatorStationName	NOT (VisibleString, according to IETF RFC 5890)	RSP-, 0xDB, 0x81, 0x01, 13

5.2.33.2.3 IOCRBlockReq

This field shall be checked according to Table 831.

Table 831 – IOCRBlockReq – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0102	—
BlockLength	!= 42 + NumberOfAPI × (8 + NumberOfIODataObjects × 6 + NumberOfIOCS × 6)	RSP-, 0xDB, 0x81, 0x02, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x02, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x02, 3
IOCRType	!= 0x01 to 0x04	RSP-, 0xDB, 0x81, 0x02, 4
IOCRReference	Is not unique OR (IOCRType == 0x4 AND no corresponding MCRBlock)	RSP-, 0xDB, 0x81, 0x02, 5
LT	Case IOCRProperties.RTClass != 0x04: != 0x8892 Case IOCRProperties.RTClass == 0x04: != 0x0800	RSP-, 0xDB, 0x81, 0x02, 6
IOCRProperties.RTClass	If ((!= 0x01 to 0x04 OR not supported in GSDML) AND ARProperties.StartupMode == Legacy) OR ((!= 0x02 to 0x04 OR not supported in GSDML) AND ARProperties.StartupMode == Advanced)	RSP-, 0xDB, 0x81, 0x02, 7
IOCRProperties.Reserved_1	!= 0x0	RSP-, 0xDB, 0x81, 0x02, 7
IOCRProperties.Reserved_2	!= 0x0	RSP-, 0xDB, 0x81, 0x02, 7
DataLength	If (IOCRProperties.RTClass == 0x04 AND DataLength != 12 to 1 440) OR ((IOCRProperties.RTClass == 0x01 OR IOCRProperties.RTClass == 0x02 OR IOCRProperties.RTClass == 0x03) AND DataLength != 40 to 1 440) OR not supported by GSDML	RSP-, 0xDB, 0x81, 0x02, 8
FrameID	If (IOCRType==(MULTICAST_CONSUMER_CR OR MULTICAST_PROVIDER_CR)) AND ((IOCRProperties.RTClass == 0x01 AND FrameID!=0xF800 to 0xFBFF) OR (IOCRProperties.RTClass == 0x02 AND FrameID!=0xBC00 to 0xBFFF) OR (IOCRProperties.RTClass == 0x03 AND FrameID!=0x0100 to 0x7FFF) OR (IOCRProperties.RTClass == 0x04 AND FrameID!=0xF800 to 0xFBFF))	RSP-, 0xDB, 0x81, 0x02, 9

Parameter	Checking rules	Behavior on match
FrameID	If IOCRTYPE==INPUT_CR AND ((IOCProperties.RTClass == 0x01 AND FrameID!=0xC000 to 0xF7FF) OR (IOCProperties.RTClass == 0x02 AND FrameID!=0x8000 to 0xBBFF) OR (IOCProperties.RTClass == 0x03 AND FrameID!=0x0100 to 0x7FFF) OR (IOCProperties.RTClass == 0x04 AND FrameID!=0xC000 to 0xF7FF))	RSP-, 0xDB, 0x81, 0x02, 9
SendClockFactor	!=1 to 128 OR not supported in GSDML OR (SendClockFactor != LocalActiveSendClockFactor)^a ^a Special case for devices supporting local adaptation (optional): Replaced by "Not representable combination of SendClockFactor and ReductionRatio compared to the LocalActiveSendClockFactor" //NOTE LocalActiveSendClockFactor is given by an already established AR	RSP-, 0xDB, 0x81, 0x02, 10
ReductionRatio	If IOCProperties.RTClass == 0x01 to 0x03 AND ((ReductionRatio != 1 to 512 OR not supported in GSDML) OR (ReductionRatio ≥ 256 AND SendClockFactor > 64) OR (ReductionRatio == 512 AND SendClockFactor > 32) OR MinDeviceInterval > SendClockFactor × ReductionRatio)	RSP-, 0xDB, 0x81, 0x02, 11
ReductionRatio	If IOCProperties.RTClass == 0x04 AND ((ReductionRatio != 1 to 16 384 OR not supported in GSDML) OR (ReductionRatio ≥ 8 192 AND SendClockFactor > 64) OR (ReductionRatio == 16 384 AND SendClockFactor > 32))	RSP-, 0xDB, 0x81, 0x02, 11
Phase	If Phase == 0x0 OR Phase > ReductionRatio	RSP-, 0xDB, 0x81, 0x02, 12

Parameter	Checking rules	Behavior on match
Sequence	not checked	—
FrameSendOffset	If (FrameSendOffset $\geq$ SendClockFactor $\times$ 31 250 ns) OR ((FrameSendOffset $\geq$ 0x003D0900) AND (FrameSendOffset $\leq$ 0xFFFFFFFF))	RSP-, 0xDB, 0x81, 0x02, 14
DataHoldFactor (legacy)	if DataHoldFactor (legacy) $\neq$ DataHoldFactor	RSP-, 0xDB, 0x81, 0x02, 15
DataHoldFactor	If DataHoldFactor $\neq$ 0x0001 to 0x1E00 OR (IOCRProperties.RTClass == 0x04 AND DataHoldFactor $\times$ ReductionRatio $\times$ SendClockFactor $\times$ 31,25 > 61 440 000) OR (IOCRProperties.RTClass == 0x01 to 0x03 AND DataHoldFactor $\times$ ReductionRatio $\times$ SendClockFactor $\times$ 31,25 > 1 920 000)	RSP-, 0xDB, 0x81, 0x02, 16
IOCRTagHeader.IOCRVLANID	$\neq$ No VLAN	RSP-, 0xDB, 0x81, 0x02, 17
IOCRTagHeader.IOUserPriority	$\neq$ IO CR Priority	RSP-, 0xDB, 0x81, 0x02, 17
IOCRMulticastMACAdd	Case IOCRTYPE == 0x03 or 0x04: Is not multicast address	RSP-, 0xDB, 0x81, 0x02, 18
NumberOfAPI	NumberOfAPI == 0	RSP-, 0xDB, 0x81, 0x02, 19
API	API not in corresponding ExpectedSubmoduleBlockReq	RSP-, 0xDB, 0x81, 0x02, 20
NumberOfIODataObjects	== 0x0 AND NumberOfIOCS = 0x0	RSP-, 0xDB, 0x81, 0x02, 21
SlotNumber	Not in corresponding ExpectedSubmoduleBlockReq	RSP-, 0xDB, 0x81, 0x02, 22
SubslotNumber	Is not unique (in conjunction with SlotNumber) OR Not in corresponding ExpectedSubmoduleBlockReq OR ((corresponding SubmoduleProperties.Type == 0x0 to 0x1) AND (IOCRTYPE == 0x2 OR 0x4)) OR ((corresponding SubmoduleProperties.Type == 0x2) AND (IOCRTYPE == 0x1 OR 0x3))	RSP-, 0xDB, 0x81, 0x02, 23

Parameter	Checking rules	Behavior on match
IODataObjectFrameOffset	Case “SubmoduleProperties.ReduceInputSubmoduleDataLength” OR “SubmoduleProperties.ReduceOutputSubmoduleDataLength” smLength:= 0 psLength:= LengthIOPS Case “SubmoduleProperties.DiscardIOXS” smLength:= 0 psLength:= 0 Default smLength:= SubmoduleDataLength psLength:= LengthIOPS If ((IODataObjectFrameOffset + (smLength + psLength)) > DataLength) OR ((SubmoduleProperties.DiscardIOXS == FALSE) AND “occupied octets” overlap with other IODataObjects or IOCSes) OR not supported in GSDML	RSP-, 0xDB, 0x81, 0x02, 24
NumberOfIOCS	== 0x0 AND NumberOfIODataObjects == 0x0	RSP-, 0xDB, 0x81, 0x02, 25
SlotNumber	Not in corresponding ExpectedSubmoduleBlockReq	RSP-, 0xDB, 0x81, 0x02, 26
SubslotNumber	Is not unique (in conjunction with SlotNumber) OR Not in corresponding ExpectedSubmoduleBlockReq OR ((corresponding SubmoduleProperties.Type == 0x0 to 0x1) AND (IOCRTType == 0x1)) OR ((corresponding SubmoduleProperties.Type == 0x2) AND (IOCRTType == 0x2))	RSP-, 0xDB, 0x81, 0x02, 27
IOCSFrameOffset	Case “SubmoduleProperties.DiscardIOXS” (IOCSFrameOffset + 0 > DataLength) Default (IOCSFrameOffset + LengthIOCS) > DataLength) OR “occupied octets” overlap with other IODataObjects or IOCSes	RSP-, 0xDB, 0x81, 0x02, 28

5.2.33.2.4 AlarmCRBlockReq

This field shall be checked according to Table 832.

Table 832 – AlarmCRBlockReq – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0103	—
BlockLength	!= 22	RSP-, 0xDB, 0x81, 0x04, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x04, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x04, 3
AlarmCRType	!= 0x0001	RSP-, 0xDB, 0x81, 0x04, 4
LT	Case AlarmCRProperties.Transport == 0x00: != 0x8892 Case AlarmCRProperties.Transport == 0x01: != 0x0800	RSP-, 0xDB, 0x81, 0x04, 5
AlarmCRProperties.Priority	Not checked	—
AlarmCRProperties.Transport	Not checked	—
AlarmCRProperties.Reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x04, 6
RTATimeoutFactor	!= 0x0001 to 0x0064 optional: AND != 0x0065 to 0xFFFF	RSP-, 0xDB, 0x81, 0x04, 7
RTARetries	!= 3 to 15	RSP-, 0xDB, 0x81, 0x04, 8
LocalAlarmReference	Not checked	—
MaxAlarmDataLength	!= 200 to 1 432	RSP-, 0xDB, 0x81, 0x04, 10
AlarmCRTagHeaderHigh. AlarmCRVLANID	!= No VLAN	RSP-, 0xDB, 0x81, 0x04, 11
AlarmCRTagHeaderHigh. AlarmUserPriority	!= Alarm CR Priority High	RSP-, 0xDB, 0x81, 0x04, 11
AlarmCRTagHeaderLow. AlarmCRVLANID	!= No VLAN	RSP-, 0xDB, 0x81, 0x04, 12
AlarmCRTagHeaderLow. AlarmUserPriority	!= Alarm CR Priority Low	RSP-, 0xDB, 0x81, 0x04, 12

5.2.33.2.5 ExpectedSubmoduleBlockReq

This field shall be checked according to Table 833.

Table 833 – ExpectedSubmoduleBlockReq – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0104	—
BlockLength	!= 4 + NumberOfAPI × (14 + NumberOfSubmodules × (8 + NumberOfSubsequentDataDescriptionBlocks × 6)	RSP-, 0xDB, 0x81, 0x03, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x03, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x03, 3
NumberOfAPI	== 0	RSP-, 0xDB, 0x81, 0x03, 4
API	> 0x0 AND not supported in GSDML	RSP-, 0xDB, 0x81, 0x03, 5
SlotNumber	!= 0x0 to 0x7FFF OR not supported in GSDML OR not unique	RSP-, 0xDB, 0x81, 0x03, 6

Parameter	Checking rules	Behavior on match
ModuleIdentNumber	== 0x00000000	RSP-, 0xDB, 0x81, 0x03, 7
ModuleProperties.Reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x03, 8
NumberOfSubmodules	== 0	RSP-, 0xDB, 0x81, 0x03, 9
SubslotNumber	!= 0x0000 to 0x9FFF OR not supported in GSDML OR not unique OR not used in at least one IOCR OR == 0x8000 to 0x9FFF AND API != 0	RSP-, 0xDB, 0x81, 0x03, 10
SubmoduleProperties.Type	== ((NO_IO OR INPUT) AND !(input description block follows)) OR == ((OUTPUT) AND !(output description block follows)) OR == (INPUT_OUTPUT) AND !(input description block follows AND output description block follows)	RSP-, 0xDB, 0x81, 0x03, 12
SubmoduleProperties.SharedInput	If SubmoduleProperties.Type == OUTPUT AND SubmoduleProperties.SharedInput == 0x1	RSP-, 0xDB, 0x81, 0x03, 12
SubmoduleProperties.ReduceInputSubmoduleDataLength	If (SubmoduleProperties.ReduceInputSubmoduleDataLength == TRUE AND SubmoduleProperties.Type != (INPUT OR INPUT_OUTPUT))	RSP-, 0xDB, 0x81, 0x03, 12
SubmoduleProperties.ReduceOutputSubmoduleDataLength	If (SubmoduleProperties.ReduceOutputSubmoduleDataLength == TRUE AND SubmoduleProperties.Type != (OUTPUT OR INPUT_OUTPUT))	RSP-, 0xDB, 0x81, 0x03, 12
SubmoduleProperties.DiscardIOXS	If (SubmoduleProperties.DiscardIOXS == TRUE AND SubmoduleProperties.Type != NO_IO)	RSP-, 0xDB, 0x81, 0x03, 12
SubmoduleProperties.reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x03, 12
DataDescription.Type	If DataDescription.Type == (0x00 OR 0x03) OR (DataDescription.Type == 0x01 AND SubmoduleProperties.Type == 0x02) OR (DataDescription.Type == 0x02 AND SubmoduleProperties.Type != (0x02 OR 0x03))	RSP-, 0xDB, 0x81, 0x03, 13
DataDescription.Reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x03, 13
SubmoduleDataLength	If SubmoduleDataLength != 0 to 1 439 OR (SubmoduleProperties.Type == NO_IO AND SubmoduleDataLength != 0)	RSP-, 0xDB, 0x81, 0x03, 14
LengthIOPS	!= 0x01	RSP-, 0xDB, 0x81, 0x03, 15
LengthIOCS	!= 0x01	RSP-, 0xDB, 0x81, 0x03, 16

5.2.33.2.6 PrmServerBlock

This field shall be checked according to Table 834.

Table 834 – PrmServerBlock – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0105	—
BlockLength	!= 26 + StationNameLength	RSP-, 0xDB, 0x81, 0x05, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x05, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x05, 3
ParameterServerObjectUUID	Not checked	—
ParameterServerProperties	Not checked	—
CMIInitiatorActivityTimeoutFactor	!= 1 to 1 000	RSP-, 0xDB, 0x81, 0x05, 6
StationNameLength	!= 1 to 240	RSP-, 0xDB, 0x81, 0x05, 7
ParameterServerStationName	Not visible string according to IETF RFC 5890	RSP-, 0xDB, 0x81, 0x05, 8

5.2.33.2.7 MCRBlock

This field shall be checked according to Table 835.

Table 835 – MCRBlockReq – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0106	—
BlockLength	!= (12 + StationNameLength + Padding) AND (BlockLength mod 4 == 0)	RSP-, 0xDB, 0x81, 0x06, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x06, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x06, 3
IOCRReference	Is not unique OR Corresponding IOCRTYPE != 0x4	RSP-, 0xDB, 0x81, 0x06, 4
AddressResolutionProperties. Protocol	!= 0x01 OR 0x02	RSP-, 0xDB, 0x81, 0x06, 5
AddressResolutionProperties. Reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x06, 5
AddressResolutionProperties. Factor	!= 0x0001 to 0x0064 optional: AND != 0x0065 to 0xFFFF	RSP-, 0xDB, 0x81, 0x06, 5
MCITimeoutFactor	> 100	RSP-, 0xDB, 0x81, 0x06, 6
StationNameLength	!= 1 to 240	RSP-, 0xDB, 0x81, 0x06, 7
ProviderStationName	Not visible string according to IETF RFC 5890	RSP-, 0xDB, 0x81, 0x06, 8
Padding	Not checked	—

5.2.33.2.8 ARRPCBlockReq

This field shall be checked according to Table 836.

Table 836 – ARRPCBlockReq – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0107	—
BlockLength	!= 4	RSP-, 0xDB, 0x81, 0x07, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x07, 2
BlockVersionLow	!= 0x01	RSP-, 0xDB, 0x81, 0x07, 3
InitiatorRPCServerPort	!= 0x0400 to 0xFFFF	RSP-, 0xDB, 0x81, 0x07, 4

5.2.33.2.9 ARVendorBlockReq

5.2.33.2.9.1 General

This field shall be checked regarding definitions to BlockType 0x0108. No detailed checking rules are defined.

5.2.33.2.9.2 Extended identification rules

The “Extended identification rules” extends the identification scheme of this document. Thus this information is treated as an additional definition space for ModuleIdentNumber and SubmoduleIdentNumber. The check of the semantics of the ExpectedExtendedIdentification-Info may lead to “Wrong Module” or “Wrong Submodule” and thus a ModuleDiffBlock.

5.2.33.2.10 IRInfoBlock

This field shall be checked according to Table 837.

Table 837 – IRInfoBlock – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0109	—
BlockLength	!= 1 + 1 + 2(Padding) + 16 + 2(Padding) + 2 + NumberOfIOCRs * (2 + 2 + 4)	RSP-, 0xDB, 0x81, 0x09, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x09, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x09, 3
Padding	Not checked	—
IRDataUUID	== 0	RSP-, 0xDB, 0x81, 0x09, 5
Padding	Not checked	—
NumberOfIOCRs	!= (0 OR 2 OR not supported in GSDML)	RSP-, 0xDB, 0x81, 0x09, 7
IOCRReference	no matching IOCRBlockReq or not unique	RSP-, 0xDB, 0x81, 0x09, 8
SubframeOffset	< 6 OR ≥ DataLength – (2 + 2)	RSP-, 0xDB, 0x81, 0x09, 9
SubframeData	(SubframeData.Position == 0) OR (SubframeData.DataLength == 0) OR (SubframeOffset + SubframeData.DataLength > DataLength – (2 + 2))	RSP-, 0xDB, 0x81, 0x09, 10

5.2.33.2.11 SRInfoBlock

This field shall be checked according to Table 838.

Table 838 – SRInfoBlock – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x010A	—
BlockLength	!= 2 + 2 + 4	RSP-, 0xDB, 0x81, 0x0A, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x0A, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x0A, 3
RedundancyDataHoldFactor	(== 0 to 2 OR not supported in GSDML)	RSP-, 0xDB, 0x81, 0x0A, 4
SRProperties. InputValidOnBackupAR	Not checked	—
SRProperties. Mode	(== 0 to 1 OR not supported in GSDML)	RSP-, 0xDB, 0x81, 0x0A, 5
SRProperties. Reserved_1	!= 0	RSP-, 0xDB, 0x81, 0x0A, 5
SRProperties. Reserved_2	!= 0	RSP-, 0xDB, 0x81, 0x0A, 5
SRProperties. Reserved_3	Not checked	—

5.2.33.2.12 RSInfoBlock

This field shall be checked according to Table 839.

Table 839 – RSInfoBlock – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x010C	—
BlockLength	!= 2 + 2 + 4	RSP-, 0xDB, 0x81, 0x0D, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x0D, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x0D, 3
Padding	!((== 0 AND length == 2)	RSP-, 0xDB, 0x81, 0x0D, 4
RS_Properties. AlarmTransport	(!= 0 to 1 OR not supported in GSDML)	RSP-, 0xDB, 0x81, 0x0D, 5
RS_Properties. Reserved_1	!= 0	RSP-, 0xDB, 0x81, 0x0D, 5
RS_Properties. Reserved_2	Not checked	—

5.2.33.2.13 ARFSUBBlock

This field shall be checked regarding definitions to BlockType 0x010B.

5.2.33.3 IODConnectRes

5.2.33.3.1 ArgsLength

This field shall be checked according to Table 840.

Table 840 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	Okay, check IODConnectRes-PDU
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	—
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRData-Response	ArgsLength	!= Sum of all BlockLength + Number of Blocks × 4	CMCTL: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the response	CMCTL: Unknown blocks
IODConnectRes	—	—	—

5.2.33.3.2 ARBlockRes

This field shall be checked according to Table 841.

Table 841 – ARBlockRes – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8101	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 30	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARType	!= ARBlockReq.ARTYPE	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUUID	!= ARBlockReq.ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	== ARBlockReq.SessionKey	n. a.
SessionKey	!= ARBlockReq.SessionKey	ignore
CMResponderMacAdd	Is not unicast	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

Parameter	Checking rules	Behavior on match
ResponderUDPRTPort	If (IOCRProperties.RTClass == RT_CLASS_UDP) == 0 to 0x03FF else ignore	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.3.3 IOCRBlockRes

This field shall be checked according to Table 842.

Table 842 – IOCRBlockRes – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8102	n. a.
BlockLength	!= 8	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
IOCRType	!= IOCRBlockReq.IOCRType	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
IOCRReference	!= IOCRBlockReq.IOCRReference	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
FrameID	If (IOCRBlockReq.IOCRType == "Input CR" OR IOCRBlockReq.IOCRType == "Multicast Consumer CR") AND IOCRBlockReq.FrameID != FrameID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
FrameID	If (IOCRBlockReq.IOCRType == "Output CR") AND ((IOCRProperties.RTClass == 0x01 AND FrameID!=0xC000 to 0xF7FF) OR (IOCRProperties.RTClass == 0x02 AND FrameID!=0x8000 to 0xBBFF) OR (IOCRProperties.RTClass == 0x03 AND FrameID!=0x0100 to 0x7FFF AND IOCRBlockReq.FrameID != FrameID) OR (IOCRProperties.RTClass == 0x04 AND FrameID!=0xC000 to 0xF7FF))	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.3.4 AlarmCRBlockRes

This field shall be checked according to Table 843.

Table 843 – AlarmCRBlockRes – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8103	—
BlockLength	!= 8	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
AlarmCRType	!= 0x0001	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
LocalAlarmReference	Not checked	—
MaxAlarmDataLength	!= 200 to 1 432	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.3.5 ModuleDiffBlock

This field shall be checked according to Table 844.

For legacy reasons, a ModuleDiffBlock containing NumberOfAPIs == 1, API == 0 and NumberOfModules == 0, may be seen in IODConnect.rsp, ControlBlockConnect(ApiIRdy).req or ControlBlockPlug(ApiIRdy).req.

Table 844 – ModuleDiffBlock – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8104	—
BlockLength	!= 4 + NumberOfAPI × (6 + NumberOfModules × (10 + NumberOfSubmodules × 8))	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
NumberOfAPIs	== 0	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
API	!= ExpectedSubmoduleBlockReq.API	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
NumberOfModules	== 0	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SlotNumber	!= ExpectedSubmoduleBlockReq.SlotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ModuleState	== 0x0 to 0x0003	—

Parameter	Checking rules	Behavior on match
ModuleState	!= 0x0 to 0x0003	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
NumberOfSubmodules	Not checked	—
SubslotNumber	!= ExpectedSubmoduleBlockReq.SubslotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubmoduleIdentNumber	Not checked	—
SubmoduleState.AddInfo	Not checked	—
SubmoduleState.Advice	Not checked	—
SubmoduleState.Maintenance-Required	Not checked	—
SubmoduleState.Maintenance-Demanded	Not checked	—
SubmoduleState.Fault	Not checked	—
SubmoduleState.ARInfo	If SubmoduleState.FormatIndicator == 1 != 0x0 to 0x04	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubmoduleState.IdentInfo	If SubmoduleState.FormatIndicator == 1 != 0x0 to 0x03 and Table 520	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubmoduleState.FormatIndicator	Not checked	—
SubmoduleState.Detail	If SubmoduleState.FormatIndicator == 0 != (0x0 to 0x02 OR 0x04 OR 0x07)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.3.6 ARServerBlockRes

This field shall be checked according to Table 845.

Table 845 – ARServerBlockRes – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8106	—
BlockLength	!= Calculated length	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
StationNameLength	!= Length of the CMResponderStationName	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
CMResponderStationName	!= Local stored / expected CMResponderStationName	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.3.7 RRPCBlockRes

This field shall be checked regarding definitions to BlockType 0x8107. No additional checking rules are defined.

5.2.33.3.8 ARVendorBlockRes

This field shall be checked regarding definitions to BlockType 0x8108. No additional checking rules are defined.

5.2.33.4 IODControlReq

5.2.33.4.1 ArgsLength

This field shall be checked according to Table 846.

Table 846 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	Okay, check IODControlReq-PDU
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataRequest	ArgsLength	!= Sum of all BlockLength + Number of Blocks × 4	CMRPC/CMINA: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the response	CMRPC/CMINA: Unknown blocks
IODControlReq	—	—	—

5.2.33.4.2 ControlBlockConnect request

This field shall be checked according to Table 847.

Table 847 – ControlBlockConnect(PrmEnd) – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0110	RSP-, 0xDD, 0x81, 0x14, 0
BlockLength	!= 28	RSP-, 0xDD, 0x81, 0x14, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x14, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x14, 3
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x14, 4
ARUUID	!= ARBlockReq.ARUUID	CMRPC/CMINA: AR UUID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDD, 0x81, 0x14, 6
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x14, 7
ControlCommand	!= 0x0001 (PrmEnd)	RSP-, 0xDD, 0x81, 0x14, 8
ControlBlockProperties.Reserved	!= 0x0000	RSP-, 0xDD, 0x81, 0x14, 9

5.2.33.4.3 ControlBlockPlug

This field shall be checked according to Table 848.

Table 848 – ControlBlockPlug(PrmEnd) – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0111	RSP-, 0xDD, 0x81, 0x15, 0
BlockLength	!= 28	RSP-, 0xDD, 0x81, 0x15, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x15, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x15, 3
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x15, 4
ARUUID	!= ARBlockReq.ARUUID	CMRPC/CMINA: AR UUID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDD, 0x81, 0x15, 6
AlarmSequenceNumber	!= AlarmSpecifier.SequenceNumber of corresponding AlarmNotification-PDU	RSP-, 0xDD, 0x81, 0x15, 7
ControlCommand	!= 0x0001 (PrmEnd)	RSP-, 0xDD, 0x81, 0x15, 8
ControlBlockProperties.Reserved	!= 0x0000	RSP-, 0xDD, 0x81, 0x15, 9

5.2.33.4.4 ControlBlockConnect request

This field shall be checked according to Table 849.

Table 849 – ControlBlockConnect(PrmBegin) – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0118	RSP-, 0xDD, 0x81, 0x18, 0
BlockLength	!= 28	RSP-, 0xDD, 0x81, 0x18, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x18, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x18, 3
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x18, 4
ARUUID	!= ARBlockReq.ARUUID	CMRPC/CMINA: AR UUID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDD, 0x81, 0x18, 6
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x18, 7
ControlCommand	!= 0x0040 (PrmBegin)	RSP-, 0xDD, 0x81, 0x18, 8
ControlBlockProperties.Reserved	!= 0x0000	RSP-, 0xDD, 0x81, 0x18, 9

5.2.33.4.5 SubmoduleListBlock

This field shall be checked according to Table 850.

Table 850 – SubmoduleListBlock – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0119	RSP-, 0xDD, 0x81, 0x19, 0
BlockLength	!= Calculated length	RSP-, 0xDD, 0x81, 0x19, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x19, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x19, 3
NumberOfEntries	!= Fitting to the BlockLength	RSP-, 0xDD, 0x81, 0x19, 4
API	!= Fitting to the ExpectedIdentification	RSP-, 0xDD, 0x81, 0x19, 5
SlotNumber	!= Fitting to the ExpectedIdentification	RSP-, 0xDD, 0x81, 0x19, 6
SubslotNumber	!= Fitting to the ExpectedIdentification	RSP-, 0xDD, 0x81, 0x19, 7

5.2.33.5 IODControlRes**5.2.33.5.1 ArgsLength**

This field shall be checked according to Table 851.

Table 851 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	Okay, check IODControlRes-PDU
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataResponse	ArgsLength	!= Sum of all BlockLength + Number of Blocks × 4	CMCTL: ArgsLength invalid
	ArgsLength	General: If one OR more unknown blocks are contained in the response	CMCTL: Unknown blocks
IODControlRes	—	—	—

5.2.33.5.2 ControlBlockConnect

This field shall be checked according to Table 852.

Table 852 – ControlBlockConnect – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8110	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUID	!= IODControlReq.ControlBlockConnect. ARUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IODControlReq.ControlBlockConnect. SessionKey	Ignore
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.Reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.5.3 ControlBlockPlug

This field shall be checked according to Table 853.

Table 853 – ControlBlockPlug – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8111	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

Parameter	Checking rules	Behavior on match
ARUID	!= IODControlReq.ControlBlockPlug. ARUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IODControlReq.ControlBlockPlug. SessionKey	Ignore
AlarmSequenceNumber	!= AlarmSpecifier.SequenceNumber of corresponding AlarmNotification-PDU	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.Reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.5.4 ControlBlockConnect

This field shall be checked according to Table 854.

Table 854 – ControlBlockConnect(PrmBegin) – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8118	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUID	!= IODControlReq.ControlBlockConnect. PrmBegin.ARUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IODControlReq.ControlBlockConnect. PrmBegin.SessionKey	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.Reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.6 IOXControlReq**5.2.33.6.1 ArgsLength**

This field shall be checked according to Table 855.

Table 855 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	Okay, check IOXControlReq-PDU
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataRequest	ArgsLength	!= Sum of all BlockLength + Number of Blocks × 4	CMRPC / CTLRPC: ArgsLength invalid
	ArgsLength	General: If one OR more unknown blocks are contained in the request	CMRPC / CTLRPC: Unknown blocks
IOXControlReq	—	—	—

5.2.33.6.2 ControlBlockConnect

This field shall be checked according to Table 856.

Table 856 – ControlBlockConnect(AppIRdy) – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0112	RSP-, 0xDD, 0x81, 0x16, 0
BlockLength	!= 28	RSP-, 0xDD, 0x81, 0x16, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x16, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x16, 3
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x16, 4
ARUUID	!= ARBlockReq.ARUUID	CMRPC/CMINA: AR UUID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDD, 0x81, 0x16, 6
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x16, 7
ControlCommand	!= 0x0002 (ApplicationReady)	RSP-, 0xDD, 0x81, 0x16, 8
ControlBlockProperties.Reserved	!= 0x0000	RSP-, 0xDD, 0x81, 0x16, 9

5.2.33.6.3 ControlBlockPlug

This field shall be checked according to Table 857.

Table 857 – ControlBlockPlug(ApplRdy) – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0113	RSP-, 0xDD, 0x81, 0x17, 0
BlockLength	!= 28	RSP-, 0xDD, 0x81, 0x17, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x17, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x17, 3
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x17, 4
ARUID	!= ARBlockReq.ARUID	CMRPC/CMINA: AR UID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDD, 0x81, 0x17, 6
AlarmSequenceNumber	!= AlarmSpecifier.SequenceNumber of corresponding AlarmNotification-PDU	RSP-, 0xDD, 0x81, 0x17, 7
ControlCommand	!= 0x0002 (ApplicationReady)	RSP-, 0xDD, 0x81, 0x17, 8
ControlBlockProperties.Reserved	!= 0x0000	RSP-, 0xDD, 0x81, 0x17, 9

5.2.33.6.4 ExpectedSubmoduleBlockReq

This field shall be checked according to Table 833.

5.2.33.6.5 ModuleDiffBlock

This field shall be checked according to Table 844 using the ErrorCodes from Table 543 row “Faulty ModuleDiffBlock”.

5.2.33.7 IOXControlRes**5.2.33.7.1 ArgsLength**

This field shall be checked according to Table 858.

Table 858 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	Okay, check IOXControlRes-PDU
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataResponse	ArgsLength	!= Sum of all BlockLength + Number of Blocks × 4	CMRPC/CMINA: ArgsLength invalid
	ArgsLength	General: If one OR more unknown blocks are contained in the response	CMRPC/CMINA: Unknown blocks
IOXControlRes	—	—	—

5.2.33.7.2 ControlBlockConnect

This field shall be checked according to Table 859.

Table 859 – ControlBlockConnect – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8112	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUID	!= IOXControlReq.ControlBlockConnect. ARUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IOXControlReq.ControlBlockConnect. SessionKey	ignore
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.Reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.7.3 ControlBlockPlug

This field shall be checked according to Table 860.

Table 860 – ControlBlockPlug – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8113	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

Parameter	Checking rules	Behavior on match
ARUID	!= IOXControlReq.ControlBlockPlug. ARUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IOXControlReq.ControlBlockPlug. SessionKey	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
AlarmSequenceNumber	!= AlarmSpecifier.SequenceNumber of corresponding AlarmNotification-PDU	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.Reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.8 IODReleaseReq

5.2.33.8.1 ArgsLength

This field shall be checked according to Table 861.

Table 861 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	Okay, check IODReleaseReq-PDU
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	—
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataRequest	ArgsLength	!= ReleaseBlock.BlockLength + 4	CMRPC/CMINA: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the request	CMRPC/CMINA: Unknown blocks
IODReleaseReq	—	—	—

5.2.33.8.2 ReleaseBlock

This field shall be checked according to Table 862.

Table 862 – ReleaseBlock – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0114	RSP-, 0xDC, 0x81, 0x28, 0
BlockLength	!= 28	RSP-, 0xDC, 0x81, 0x28, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDC, 0x81, 0x28, 2
BlockVersionLow	!= 0x00	RSP-, 0xDC, 0x81, 0x28, 3
Padding	!= 0x00	RSP-, 0xDC, 0x81, 0x28, 4
ARUID	!= ARBlockReq.ARUID	CMRPC/CMINA: AR UID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDC, 0x81, 0x28, 6
Padding	!= 0x00	RSP-, 0xDC, 0x81, 0x28, 7
ControlCommand	!= 0x0004 (Release)	RSP-, 0xDC, 0x81, 0x28, 8
ControlBlockProperties.Reserved	!= 0x0000	RSP-, 0xDC, 0x81, 0x28, 9

5.2.33.9 IODReleaseRes**5.2.33.9.1 ArgsLength**

This field shall be checked according to Table 863.

Table 863 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	Okay, check IODReleaseRes-PDU
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	—
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataResponse	ArgsLength	!= ReleaseBlock.BlockLength + 4	CMRPC/CMINA: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the response	CMRPC/CMINA: Unknown blocks
IODReleaseRes	—	—	—

5.2.33.9.2 ReleaseBlock

This field shall be checked according to Table 864.

Table 864 – ReleaseBlock – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8114	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUUID	!= IODReleaseReq.ReleaseBlock.ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IODReleaseReq.ReleaseBlock. SessionKey	ignore
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.Reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

5.2.33.10 IODWriteReq

5.2.33.10.1 ArgsLength

This field shall be checked according to Table 865.

Table 865 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	Okay, check IODWriteReq-PDU
	RPCOperationNmb	== 4 (Control)	—
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataRequest	ArgsLength	!= (IODWriteReqHeader. BlockLength + 4 + IODWriteReqHeader. RecordDataLength)	CMRPC/CMINA: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the request	CMRPC/CMINA: Unknown blocks
IODWriteReq	—	—	—

5.2.33.10.2 IODWriteReqHeader

This field shall be checked according to Table 866.

Table 866 – IODWriteReqHeader – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0008	RSP-, 0xDF, 0x81, 0x08, 0
BlockLength	!= 60	RSP-, 0xDF, 0x81, 0x08, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDF, 0x81, 0x08, 2
BlockVersionLow	!= 0x00	RSP-, 0xDF, 0x81, 0x08, 3
SeqNumber	Not checked	—
ARUUID	!= ARBlockReq.ARUUID	CMRPC/CMINA: AR UUID unknown
API	> 0x0 AND not supported in GSDML	RSP-, 0xDF, 0x80, 0xB4, 6
SlotNumber	!= 0x0 to 0x7FFF OR not supported in GSDML	RSP-, 0xDF, 0x80, 0xB2, 7
SubslotNumber	!= 0x0 to 0x9FFF OR not supported in GSDML	RSP-, 0xDF, 0x80, 0xB2, 8
Padding	!= 0x0000	RSP-, 0xDF, 0x80, 0xB7, 9
Index	Not supported by application	RSP-, 0xDF, 0x80, 0xB0, 10 ^a
RecordDataLength	!= consistent with ArgsLength	RSP-, 0xDF, 0x81, 0x08, 11
RWPadding	Not checked	—

^a The responded error code depends on the checking sequence required for the given use case. For example the used AR is in ARstate:=BACKUP or ...

5.2.33.10.3 RecordDataWrite

This field shall be not checked.

5.2.33.11 IODWriteRes**5.2.33.11.1 ArgsLength**

This field shall be checked according to Table 867.

Table 867 – ArgsLength check

	Parameter	Checking rules	Behavior on match
NDRDataResponse	ArgsLength	!= IODWriteResHeader.BlockLength + 4	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
	—	General: If one OR more unknown blocks are contained in the response	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
IODReleaseRes	—	—	—

5.2.33.11.2 IODWriteResHeader

This field shall be checked according to Table 868.

Table 868 – IODWriteResHeader – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8008	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 60	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SeqNumber	Not checked	—
ARUID	!= IODWriteReq.IODWriteReqHeader. ARUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
API	!= IODWriteReq.IODWriteReqHeader.API	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SlotNumber	!= IODWriteReq.IODWriteReqHeader.SlotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubslotNumber	!= IODWriteReq.IODWriteReqHeader.SubslotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Index	!= IODWriteReq.IODWriteReqHeader.Index	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
RecordDataLength ^a	Not checked	—
AdditionalValue1	Not checked	—
AdditionalValue2	Not checked	—
PNIOStatus	Not checked	—
RWPPadding	Not checked	—
^a The responder should mirror the RecordDataLength from the request.		

5.2.33.12 IODWriteMultipleReq**5.2.33.12.1 ArgsLength**

This field shall be checked according to Table 869.

Table 869 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	Okay, check IODWriteMultipleReq-PDU
	RPCOperationNmb	== 4 (Control)	—
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataRequest	ArgsLength	!= IODWriteReqHeader. BlockLength + 4 + Sum of all IODWriteReqHeader.BlockLe ngth + Number of WriteBlocks × 4	CMRPC/CMINA: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the request	CMRPC/CMINA: Unknown blocks
IODWriteReq	—	—	—

5.2.33.12.2 IODWriteReqHeader (multiple)

This field shall be checked according to Table 866. The value of the parameter index shall be 0xE040.

The check of each embedded record shall be done as for an individual received IODWriteReq.

If the structure of the payload defined/identified by the parameter index= 0xE040 is broken, an ErrorCode “WRITE”, ErrorDecode “PMIO”, and ErrorCode1 “Read Write Record Parameter Error” shall be responded to.

5.2.33.12.3 IODWriteReqHeader

This field shall be checked according to Table 866.

5.2.33.12.4 RecordDataWrite

This field shall contain the structure and the values defined for the parameter index stated in the IODWriteReqHeader.

5.2.33.12.5 Padding

The number of padding octets shall be 0, 1, 2 and 3 to have 32 bit alignment to the next IODWriteReqHeader. The value of the padding octets is not checked.

5.2.33.13 IODWriteMultipleRes**5.2.33.13.1 ArgsLength**

This field shall be checked according to Table 870.

Table 870 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—
	RPCOperationNmb	== 3 (Write)	Okay, check IODWriteMultipleRes-PDU
	RPCOperationNmb	== 4 (Control)	—
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRData-Response	ArgsLength	!= IODWriteResHeader.BlockLength + 4 + Sum of all IODWriteResHeader.BlockLength + Number of IODWriteResHeaderBlocks × 4	CMRPC/CMINA: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the response	CMRPC/CMINA: Unknown blocks
IODWriteMultiple-Res	—	—	—

5.2.33.13.2 IODWriteResHeader (multiple)

This field shall be checked according to Table 868. The value of the parameter index shall be 0xE040.

5.2.33.13.3 IODWriteResHeader

This field shall be checked according to Table 868.

5.2.33.14 IODReadReq

5.2.33.14.1 ArgsLength

This field shall be checked according to Table 871.

Table 871 – ArgsLength check

	Parameter	Checking rules	Behavior on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	Okay, check IODReadReq-PDU
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	—
	RPCOperationNmb	== 5 (Read Implicit)	Okay, check IODReadReq-PDU
NDRDataRequest	ArgsLength	!= IODReadReqHeader.BlockLength + 4 OR != IODReadReqHeader.BlockLength + 4 + RecordDataReadQuery.BlockLength + 4	CMRPC/CMINA: ArgsLength invalid

	Parameter	Checking rules	Behavior on match
	—	General: If one OR more unknown blocks are contained in the request	CMRPC/CMINA: Unknown blocks
IOReadReq	—	—	—

5.2.33.14.2 IODReadReqHeader

This field shall be checked according to Table 872.

Table 872 – IODReadReqHeader – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0009	RSP-, 0xDE, 0x81, 0x08, 0
BlockLength	!= 60	RSP-, 0xDE, 0x81, 0x08, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDE, 0x81, 0x08, 2
BlockVersionLow	!= 0x00	RSP-, 0xDE, 0x81, 0x08, 3
SeqNumber	Not checked	—
ARUUID	Case RPCOperationNmb == 2 (Read): != ARBlockReq.ARUUID	CMRPC/CMINA: AR UUID unknown
ARUUID	Case RPCOperationNmb == 5 (Read Implicit): != NIL	RSP-, 0xDE, 0x81, 0x08, 5
API	> 0x0 AND not supported in GSDML	RSP-, 0xDE, 0x80, 0xB4, 6
SlotNumber	!= 0x0 to 0x7FFF OR not supported in GSDML	RSP-, 0xDE, 0x80, 0xB2, 7
SubslotNumber	!= 0x0 to 0x9FFF OR not supported in GSDML	RSP-, 0xDE, 0x80, 0xB2, 8
Padding	!= 0x0000	RSP-, 0xDE, 0x80, 0xB7, 9
Index	Not supported by application	RSP-, 0xDE, 0x80, 0xB0, 10
RecordDataLength	Not consistent with ArgsMaximum	RSP-, 0xDE, 0x81, 0x08, 11
TargetARUUID	Case RPCOperationNmb == 5 (Read Implicit) AND ("ARUUID context" is "Needed" for Index): If no AR with ARUUID == TargetARUUID exist	RSP-, 0xDE, 0x81, 0x08, 12 ^a
RWPadding	Not checked	—

^a IEC 61158-5-10 allows additional the usage of "RSP-, 0xDE, 0x80, 0xB8, xx" in this case.

5.2.33.14.3 RecordDataReadQuery

This field shall be checked according to Table 873.

Table 873 – RecordDataReadQuery – request check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x0500 OR not supported in GSDML	CMRPC/CMINA: Unknown blocks
BlockLength	< 0x3 OR not consistent with ArgsLength	RSP-, 0xDE, 0x81, 0x08, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDE, 0x81, 0x08, 2
BlockVersionLow	!= 0x00	RSP-, 0xDE, 0x81, 0x08, 3
Data	Not checked	—

5.2.33.15 IODReadRes

5.2.33.15.1 ArgsLength

This field shall be checked according to Table 874.

Table 874 – ArgsLength check

	Parameter	Checking rules	Behavior on match
NDRDataResponse	ArgsLength	!= IODReadResHeader.BlockLength + 4 + IODReadResHeader.RecordDataLength	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
	—	General: If one OR more unknown blocks are contained in the response	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
IODReadRes	—	—	—

5.2.33.15.2 IODReadResHeader

This field shall be checked according to Table 875.

Table 875 – IODReadResHeader – response check

Parameter	Checking rules	Behavior on match
BlockType	!= 0x8009	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 60	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SeqNumber	Not checked	—
ARUUID	!= IODReadReqHeader.ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

Parameter	Checking rules	Behavior on match
API	$\neq$ IODReadReqHeader.API	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SlotNumber	$\neq$ IODReadReqHeader.SlotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubslotNumber	$\neq$ IODReadReqHeader.SubslotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	$\neq$ 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Index	$\neq$ IODReadReqHeader.Index	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
RecordDataLength	Not consistent with ArgsLength	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
AdditionalValue1	Not checked	—
AdditionalValue2	Not checked	—
RWPadding	Not checked	—

5.2.33.15.3 RecordDataRead

This field shall be not checked.

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 66. The general structure is according to the IEC 61158-6 series protocol machine model.

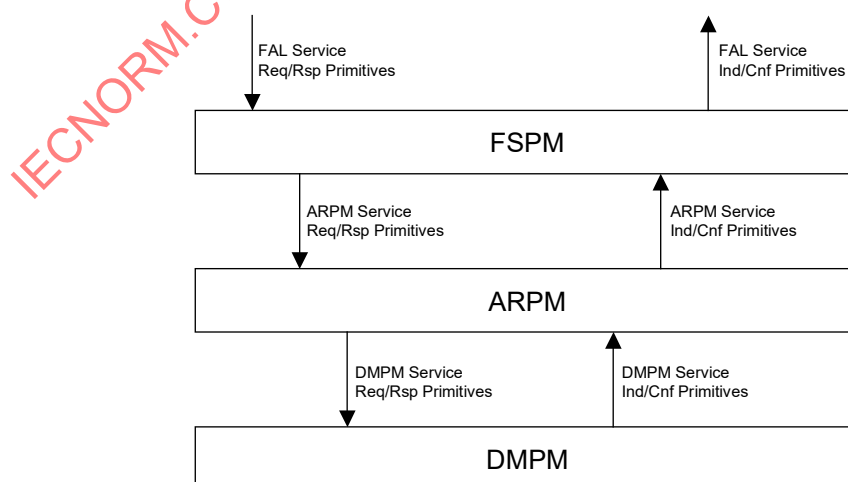


Figure 127 – Relationship among Protocol Machines

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 defines therefore two protocol machines, the LMPM and the MAC protocol machines.

5.3.1.2 FAL Service Protocol Machines

The FSPM State Machines co-ordinate 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 PPM, CPM, ALPMI, ALPMR, APMR, APMS, CMDEV, CMCTL, CMRPC, CMINA, CTRLRPC, CTLDINA State Machines 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 PPM, CPM, ALPMI, ALPMR, APMR, APMS, CMDEV, CMCTL, CMRPC, CMINA, CTRLRPC, CTLDINA State Machines 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 DLL Mapping Protocol Machines

The DL Mapping Protocol Machines (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 directly put upon the DLL User – DLL interface as described in Data Link Layer Service Definitions (Refer to Enhanced Internal Sublayer Services IEEE 802.3).

The DMPM uses for the mapping of the DMPM functions on the Layer 2 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 two kinds of FSPM defined, one for the IO device (FSPMDEV) and one for the IO controller (FSPMCTL).

5.5.2 FAL Service Protocol Machine Device

5.5.2.1 Overview

The FSPMDEV is responsible for the startup of the underlying protocol machines (LMPM, LLDP, DHCP/DCP, IP, UDP, RPC, CMDEV, CMCTL, DCP, PTCP, ALPM, PPM, CPM...).

5.5.2.2 Primitive definitions

Table 876 shows the service primitives including their associated parameters issued by the AP-Context (FAL user) and received by the FSPMDEV with the mapping to underlying services.

Table 876 – 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	PPM_Set_Data_req (CREP,	MAP_SIN_REQ

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
	Subslot Number IOPS Subslot Input Data	Data)	
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-
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	CM_Read.rsp (-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1,	MAP_READ_RSP-

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
	Add Data 2	AddData2)	
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+
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 877 shows the service primitives including their associated parameters issued by the FSPMDEV and received by the AP-Context (FAL user).

Table 877 – 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
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 (-) (	MAP_SIN_CNF	local Set	AREP

Primitive mapped from	Parameter mapped from	Primitive name	Associated parameters
CREP, ERRCLS, ERRCODE)		Input.cnf (-)	
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
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	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

Primitive mapped from	Parameter mapped from	Primitive name	Associated parameters
ARFSUBBlock)			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, TargetARUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_IND	Read.ind	AREP API Target ARUID Slot Number Subslot Number Index Seq Number Length
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

5.5.2.3 State transition diagram

The FSPMDEV defines no state machine. Therefore no states are described.

5.5.2.4 State machine description

The FSPMDEV defines no state machine. Therefore no state machine description exists.

5.5.2.5 FSPMDEV state table

The FSPMDEV defines no state machine because all services are transferred to the underlying protocol machines.

5.5.2.6 Functions, Macros, Timers and Variables

Table 878 shows the functions, macros, timers and variables defined for FSPMDEV, which are used by service primitives issued by the FAL user.

Table 878 – 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 (=)

Name	Function
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 879 shows the functions, macros, timers and variables defined for FSPMDEV, which are used by services primitives issued by the FSPMDEV.

Table 879 – 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

Name	Function
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
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

Name	Function
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
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

5.5.3 FAL Service Protocol Machine Controller

5.5.3.1 Overview

The FSPMCTL is responsible for the startup of the underlying protocol machines (LMPM, LLDP, DHCP/DCP, IP, UDP, RPC, CMDEV, CMCTL, DCP, PTCP, ALPM, PPM, CPM...).

5.5.3.2 Primitive definitions

Table 880 shows the service primitives including their associated parameters issued by the AP-Context (FAL user) and received by the FSPMCTL with the mapping to underlying services.

Table 880 – 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
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_RES-
Application Ready.rsp(+)	AREP Session Key Alarm Sequence Number	CM_Ccontrol.rsp (+)(AREP, ControlBlock)	MAP_AREADY_RES+
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	CM_Connect.req (AREP ARBlockReq ListOfIOCRBlockReq ListOfExpectedSubmoduleBlockReq AlarmCRBlockReq ParameterServerBlock ListOfMulticastCRBlock ARRPCBlock IRInfosBlock SRInfoBlock ARVendorBlock ARServerBlock ARFSUBlock)	MAP_CON_REQ

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
	AR Vendor Block AR Server Block AR FSU Block		
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-
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	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

Primitive name	Associated parameters	Primitive mapped to	Parameter mapped to
	Alarm Item		
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 881 shows the service primitives including their associated parameters issued by the FSPMCTL and received by the AP-Context (FAL user).

Table 881 – 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 (+) (	MAP_GOUTIOCS_CNF+	local Get Output	AREP

Primitive mapped to	Parameter mapped from	Primitive name	Associated parameters
CREP, Status, RecvCounter)		IOCS.cnf(+)	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
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

Primitive mapped to	Parameter mapped from	Primitive name	Associated parameters
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
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,	MAP_WRITE_CNF-	Write.cnf(-)	AREP Seq Number Error Decode Error Code 1

Primitive mapped to	Parameter mapped from	Primitive name	Associated parameters
ErrorCode2, AddData1, AddData2)			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
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 FSPMCTL defines no state machine. Therefore no states are described.

5.5.3.4 State machine description

The FSPMCTL defines no state machine. Therefore no state machine description exists.

5.5.3.5 FSPMCTL state table

The FSPMCTL defines no state machine because all services are transferred to the underlying protocol machines.

5.5.3.6 Functions, Macros, Timers and Variables

Table 882 shows the functions, macros, timers and variables defined for FSPMCTL, which are used by services primitives issued by the FAL user.

**Table 882 – 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_RES-	AREP=AREP Error Decode Error Code 1 Error Code 2
MAP_AREADY_RES+	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 ARFSUBlock=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 883 shows the functions, macros, timers and variables defined for FSPMCTL, which are used by services primitives issued by the FSPMCTL.

Table 883 – 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_ALREADY_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

5.6 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 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 884.

Table 884 – 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.

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 885.

Table 885 – 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 128.

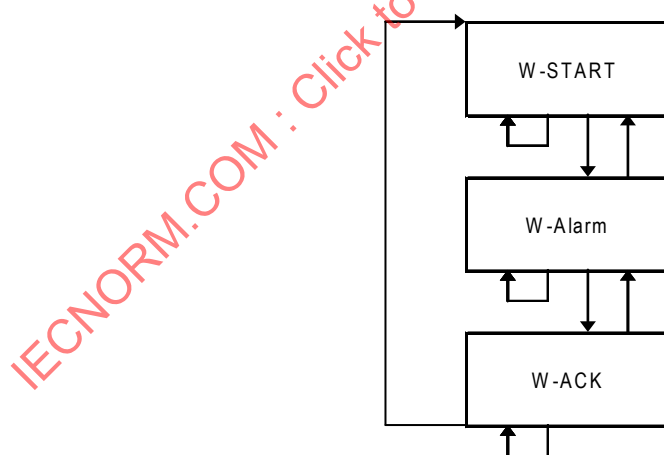


Figure 128 – State transition diagram of ALPMI

States of the ALPMI are:

- W-START** The W-START state indicates that the initialization is needed.
The Activate service sets the machine in 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 independent. 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 886 contains the description of the ALPMI state machine.

Table 886 – 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 == Alarm-Ack-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 != Alarm-Ack-PDU => ERRCLS:= ALPMI ERRCODE:= WRONG-ACK-PDU ALPMI_Error_ind ()	W-Alarm

5.6.1.5 Functions, Macros, Timers and Variables

Table 887 contains the functions, macros, timers and variables used by the ALPMI and their arguments and their descriptions.

Table 887 – 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 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 888.

Table 888 – 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_Identifier, Submodule_Identifier, User_Structure_Identifier, User_Data	The service Alarm_Notification indicates alarm data.

5.6.2.1.2 Primitives exchanged between local machines

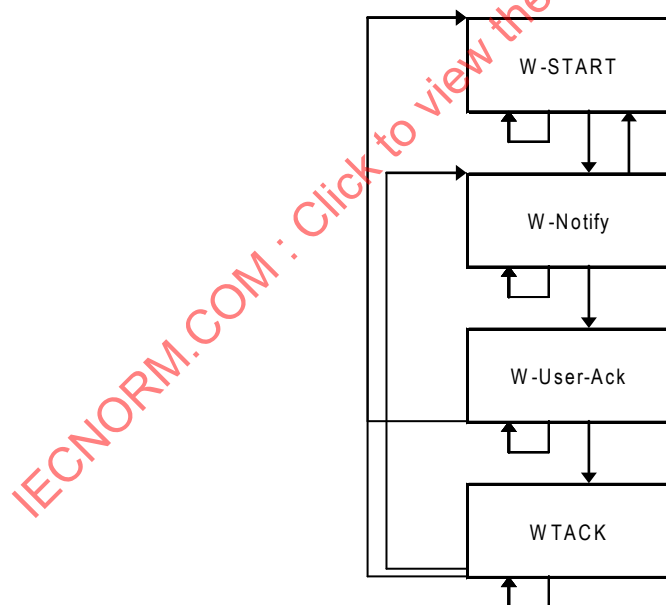
The local service primitives including their associated parameters issued or received by ALPMR are described in the service definition and shown in Table 889.

Table 889 – 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 129.

**Figure 129 – 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 in the W-Notify state.
W-Notify	After successful initialization the state machine is waiting for Alarm Notification PDUs in the state W-Notify and enters afterwards 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 independent.

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 890 contains the description of the ALPMR state machine.

Table 890 – 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 891 contains the functions, macros, timers and variables used by the ALPMR and their arguments and their descriptions.

Table 891 – 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 130 shows the integration of the IO device CM.

The following definition shall be used for ARBlockReq.ARProperties.-StartupMode == Advanced. For ARBlockReq.ARProperties.StartupMode == Legacy the definitions of the version V2.2 and the Annex B applies.

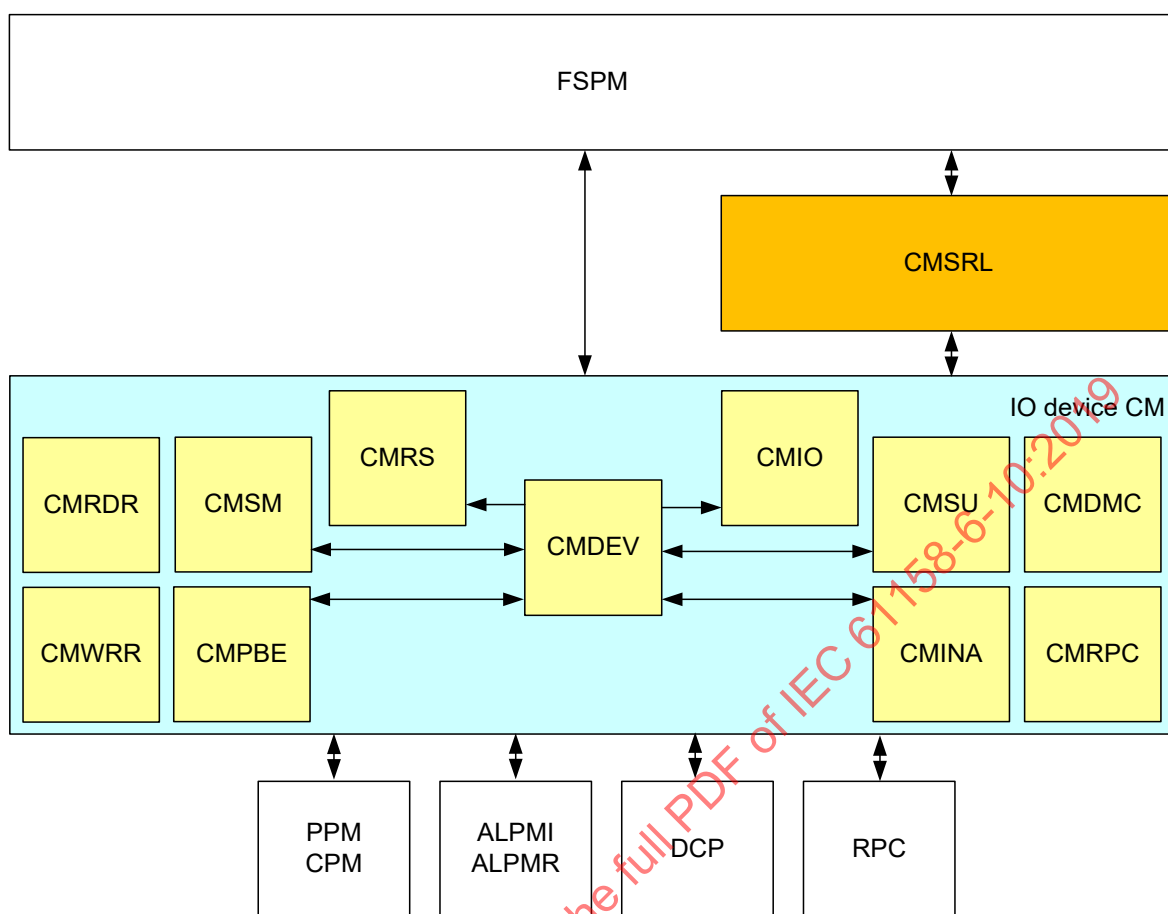


Figure 130 – Scheme of the IO device CM

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 optional state machine handles the PrmBegin, PrmEnd and ApplRdy sequence for system redundancy and dynamic reconfiguration.

CMSU

This state machine handles the startup of the different state machines during startup and shut down.

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 RPC services.

CMINA

This state machine handles IP and Name assignment.

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.

5.6.3.1.2 State transition diagram

The state transition diagram of the IO device CM is shown in Figure 131.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2019

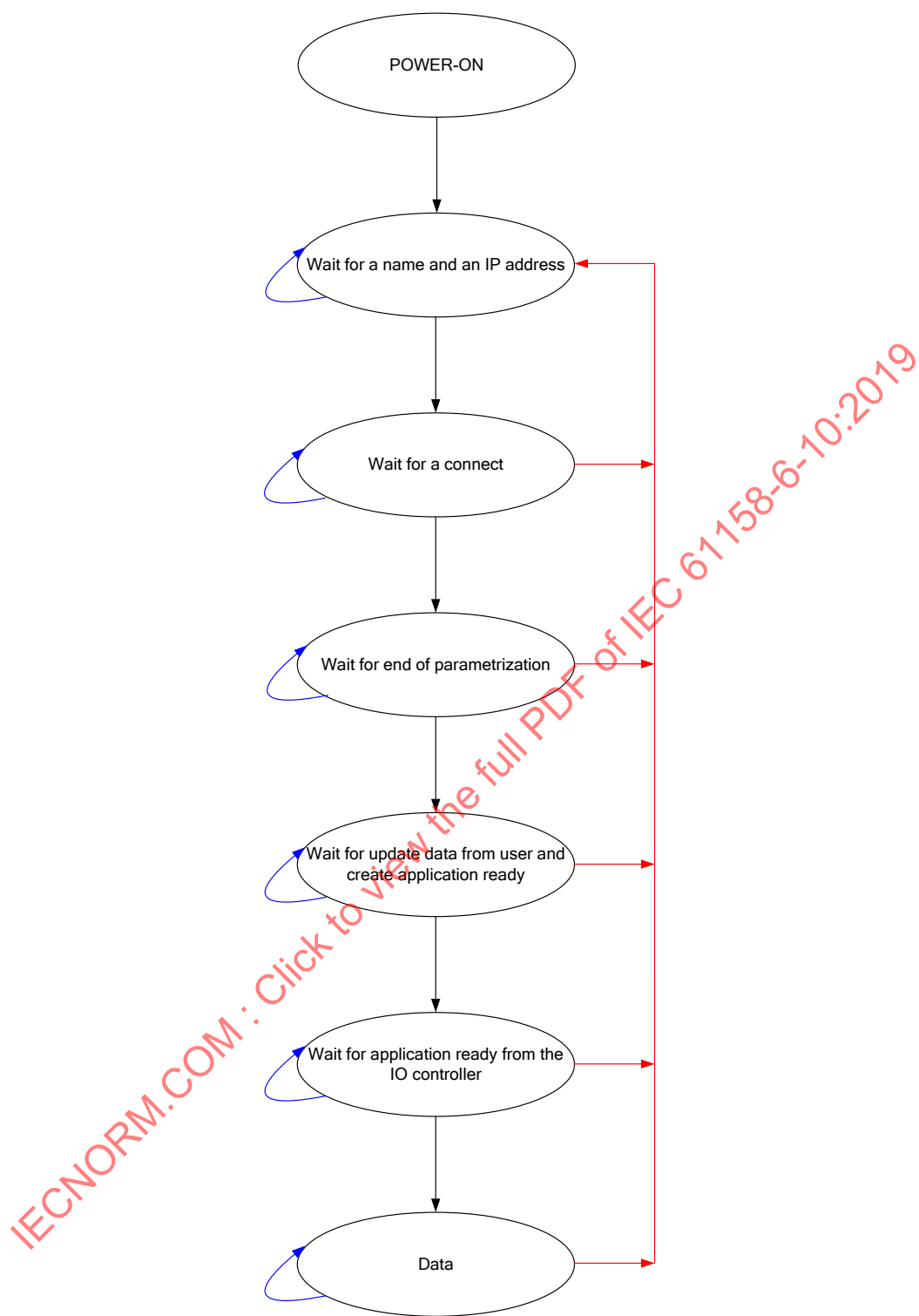


Figure 131 – State transition diagram of the IO device CM

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 executes it locally. After that, the confirmation of the application ready shall be given to the IO device.
Data	Cyclic exchange data of the submodules between the IO controller and IO devices starts, 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 service primitives including their associated parameters issued or received by Context Management Device (CMDEV) are described in the service definition and shown in Table 892.

Table 892 – 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, ErrorCode, 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 establish an application relationship.
CM_Dcontrol.rsp (-)	FSPMDEV	CMDEV	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	This service primitive is a response to end of parameter signal.
CM_Dcontrol.rsp (+)	FSPMDEV	CMDEV	AREP, ControlBlock	This service primitive is a 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 893.

Table 893 – 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.

5.6.3.2.1.2 State transition diagram

The state transition diagram of the CMDEV is shown in Figure 132.

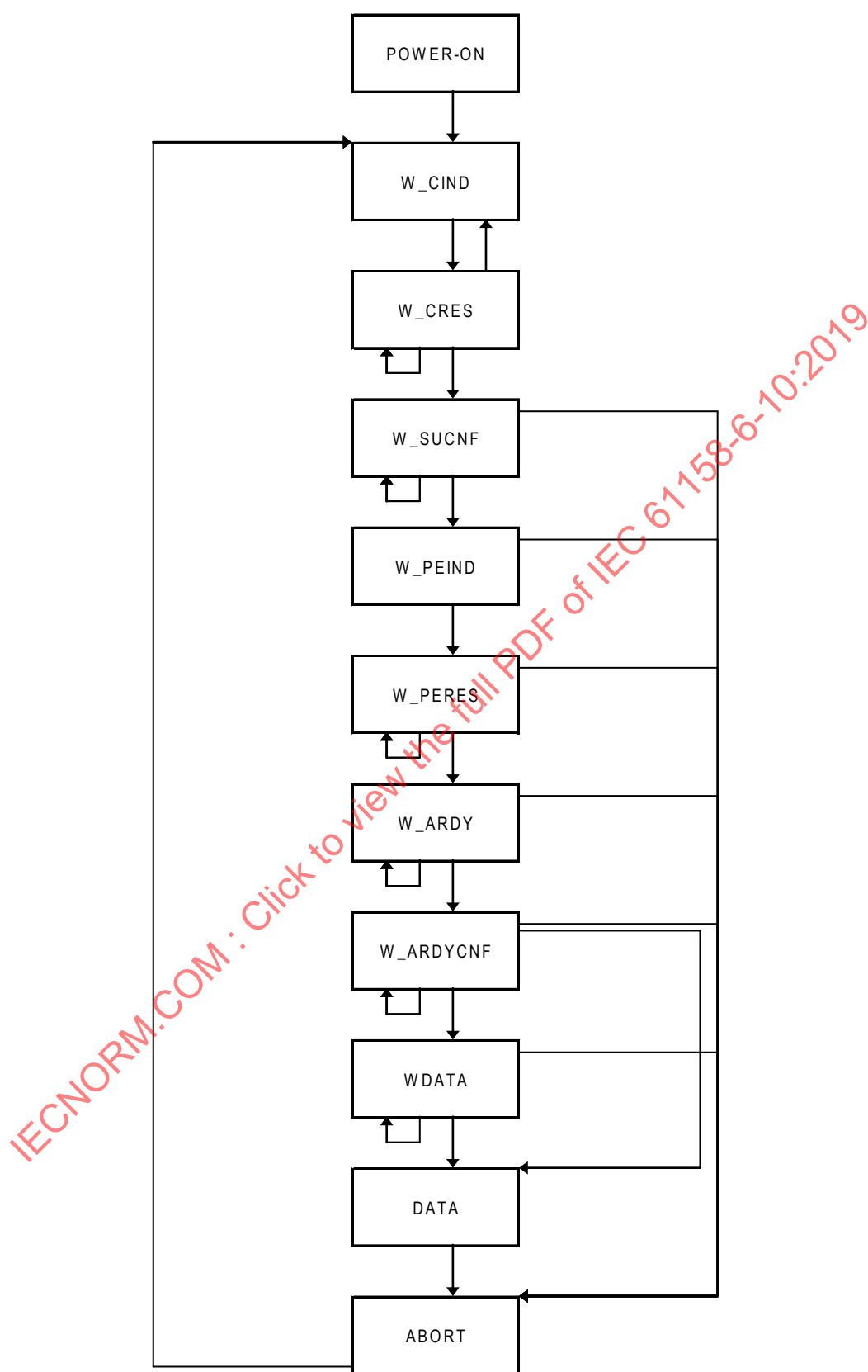


Figure 132 – State transition diagram of CMDEV

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, selected by the attribute ARProperties.DeviceAccess, handled by CMDEV and CMDEV_DA exists.

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 is not expired, the ControlBlockConnect(AppIRdy).req shall be delayed until the MCITimeoutFactor expires or if earlier, all MCRs are running.

The possible RPC reruns are handled at the CMRPC.

5.6.3.2.1.4 CMDEV state table

Table 894 contains the complete description of the CMDEV state machine.

Table 894 – CMDEV 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 => 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/duplexity" 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

5.6.3.2.1.5 Functions, Macros, Timers and Variables

Table 895 contains the functions, macros, timers and variables used by the CMDEV and their arguments and their descriptions.

Table 895 – Functions, Macros, Timers and Variables used by CMDEV

Name	Type	Function/Meaning
CheckAPDU	Function	This local function checks the semantics of the connect.
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" duplexity, after the parameterization is applied.
All PPM buffer are updated and valid	Macro	This local macro return 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 133 shows the integration of the IO device CM – device access.

The following definition shall be used for ARBlockReq.ARProperties.StartupMode == Advanced. For ARBlockReq.ARProperties.StartupMode == Legacy the definitions of the version V2.2 and the Annex B applies.

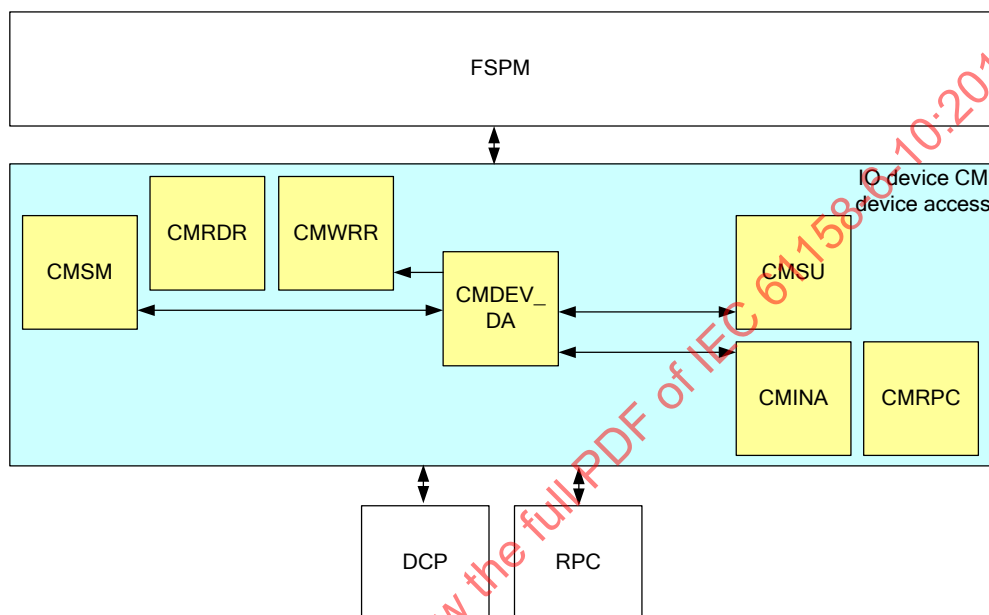


Figure 133 – Scheme of the IO device CM – device access

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

CMSU

This state machine handles the startup of the different state machines during startup and shut down

CMRPC

This machine handles the RPC communication and therefore all issued and received RPC services

CMINA

This machine handles the IP and Name assignment

5.6.3.2.2.2 Primitive definitions

5.6.3.2.2.2.1 Primitives exchanged between remote machines

The 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 896.

Table 896 – 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	—
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	—

5.6.3.2.2.2.2 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 897.

Table 897 – 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	—

5.6.3.2.2.3 State transition diagram

The state transition diagram of the CMDEV_DA is shown in Figure 132.

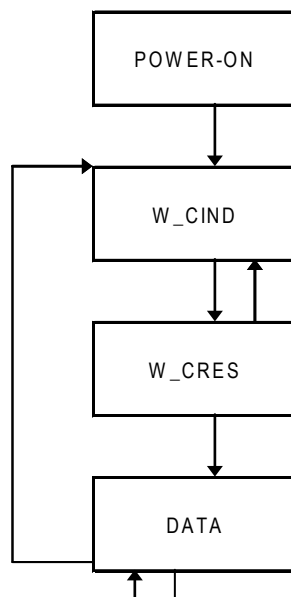


Figure 134 – State transition diagram of CMDEV_DA

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 the CMSM only STARTUP when switching to DATA to keep the monitoring using read and write services active.

5.6.3.2.2.5 CMDEV_DA (device access) state table

Table 898 contains the complete description of the CMDEV_DA state machine.

Table 898 – 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 899 contains the functions, macros, timers and variables used by the CMDEV(DA) and their arguments and their descriptions.

Table 899 – 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 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 900.

Table 900 – 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 901.

Table 901 – 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, ListOfExpectedSubmod uleBlockReq, ListOfMCRBlockReq, ListofSubframeBlockR eq, 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.StartupM ode, ...	—
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.StartupM ode, ...	—
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 135.

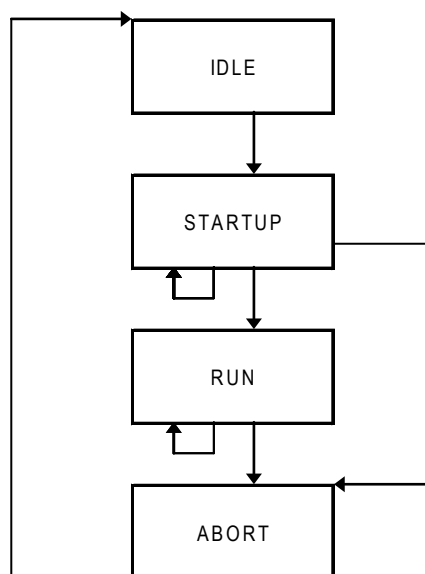


Figure 135 – State transition diagram of CMSU

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 shut down 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 receiving the connect request service and the Application ready service. The startup shall be finish before issuing the Application ready request service.

5.6.3.3.4 CMSU state table

Table 902 contains the description of the CMSU state machine. The difference between `ARProperties.StartupMode == Legacy` and `ARProperties.StartupMode == Advanced` is handled by the CMSU, the PPM and the CPM.

Table 902 – 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 903 contains the functions, macros, timers and variables used by the CMSU and their arguments and their descriptions.

Table 903 – 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 receive 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 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 904.

Table 904 – 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 905.

Table 905 – Local primitives issued or received by CMIO

Primitive	Source	Destination	Associated parameters	Functions
CMIO_info_ind	CMIO	CMDEV	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 136.

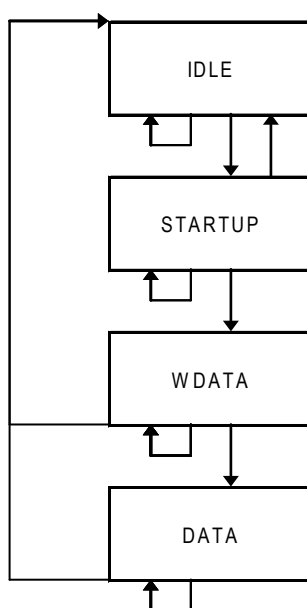


Figure 136 – State transition diagram of CMIO

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 due to all associated CPMs are receiving frames.

DATA

Data exchange phase. Monitoring the DataHoldTime.

5.6.3.4.3 State machine description

The CM Input Output protocol machine (CMIO) arranges the access to the DataStatus and NewData indication.

5.6.3.4.4 CMIO state table

Table 906 contains the complete description of the CMIO state machine.

Table 906 – 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 907 contains the functions, macros, timers and variables used by the CMIO and their arguments and their descriptions.

Table 907 – Functions used by the CMIO

Name	Type	Function/Meaning
StartTimer	Function	This local function starts or restarts a timer. The time for this timer may be 100 ms. NOTE An implementation can use “change” events instead of this timer.
StopTimer	Function	This local functions 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.

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 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 908.

Table 908 – 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 909.

Table 909 – 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 137.

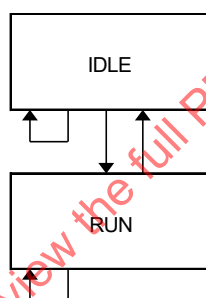


Figure 137 – State transition diagram of CMRS

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 shut down 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 receiving the Parametrization end service. The startup shall be finished before issuing the Application ready request service.

It shall be stopped when the AR is aborted.

5.6.3.5.4 CMRS state table

Table 910 contains the description of the CMRS state machine.

Table 910 – 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 911 contains the functions, macros, timers and variables used by the CMRS and their arguments and their descriptions.

Table 911 – 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 a RSInfoBlock FALSE, if the Connect service doesn't contain a 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 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 912.

Table 912 – 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 913.

Table 913 – 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 138.

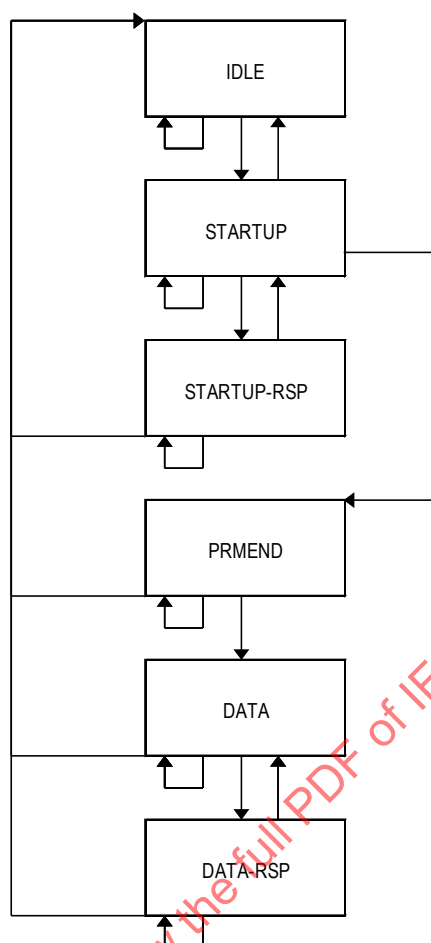


Figure 138 – State transition diagram of CMWRR

States of the CMWRR are:

IDLE	No AR context exists
STARTUP	The associated AR reaches the internal state STARTUP. Write services 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 rejected during this state.
DATA	The associated AR reaches the internal state DATA. Write services 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 914 contains the description of the CMWRR state machine. The CMWRR checks whether the ARUUID exists.

Table 914 – CMWRR state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	RM_Write.ind () => ErrorDecode:= PNORW 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:= PNORW 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
14	STARTUP-RSP	CMDEV_state_ind () /state == ABORT => ignore	IDLE
15	STARTUP-RSP	CMSRL_state_ind () => ignore	STARTUP-RSP

#	Current State	Event /Condition =>Action	Next State
16	PRMEND	RM_Write.ind () => ErrorDecode:= PNORW 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:= PNORW 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
30	DATA-RSP	CMSRL_state_ind () /ARstate == Backup => ErrorDecode:= PNORW ErrorCode1:= backup RM_Write.rsp (-)	DATA
31	DATA-RSP	CMSRL_state_ind () /ARstate != Backup => ignore	DATA-RSP

#	Current State	Event /Condition =>Action	Next State
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 915 contains the functions, macros, timers and variables used by the CMWRR and their arguments and their descriptions.

Table 915 – 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 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 916.

Table 916 – Remote primitives issued or received by CMRDR

Primitive	Source	Destination	Associated parameters	Functions
CM_Read.rsp (-)	FSPMDEV	CMRDR	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
CM_Read.rsp (+)	FSPMDEV	CMRDR	AREP, SeqNumber, Length, Data	—
RM_Read.ind	CMRPC	CMRDR	AREP, API, TargetUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length	—
CM_Read.ind	CMRDR	FSPMDEV	AREP, API, TargetUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length	—
RM_Read.rsp (-)	CMRDR	CMRPC	AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2	—
RM_Read.rsp (+)	CMRDR	CMRPC	AREP, SeqNumber, Length, Data	—

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 917.

Table 917 – 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 139.

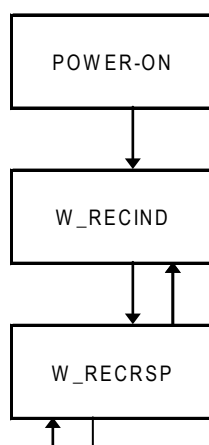


Figure 139 – State transition diagram of CMRDR

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 918 contains the complete description of the CMRDR state machine. The CMRPC checks whether the ARUUID exists.

Table 918 – CMRDR state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> Init	W_RECIND
2	W_RECIND	RM_Read.ind () => CM_Read.ind ()	W_RECRSP
3	W_RECRSP	CM_Read.rsp (+) => RM_Read.rsp (+)	W_RECIND
4	W_RECRSP	CM_Read.rsp (-) => RM_Read.rsp (-)	W_RECIND
5	W_RECRSP	RM_Read.ind () => ignore	W_RECRSP

The serialization of the “Read” RPC calls for one AR is done by the RPC 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 919 contains the functions, macros, timers and variables used by the CMRDR and their arguments and their descriptions.

Table 919 – 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 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 920.

Table 920 – 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 921.

Table 921 – 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 "PRMEND" start CMI timeout State "DATA" start CMI timeout

5.6.3.8.2 State transition diagram

The state transition diagram of the CMSM is shown in Figure 140.

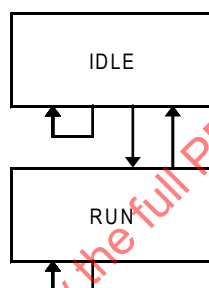


Figure 140 – State transition diagram of CMSM

States of the CMSM are:

IDLE	Wait for the start of the connection monitoring
RUN	Connection monitoring running

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 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 922 contains the complete description of the CMSM state machine.

Table 922 – CMSM state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	CMDEV_state_ind () /state == STARTUP => StartTimer (CMI, CMIInitiatorActivityTimeout)	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, CMIInitiatorActivityTimeout) RM_Read.rsp (+)	RUN
14	RUN	CM_Read.rsp () => StartTimer (CMI, CMIInitiatorActivityTimeout)	RUN
15	RUN	CM_Write.rsp () => StartTimer (CMI, CMIInitiatorActivityTimeout)	RUN
16	RUN	CM_Read.ind ()	RUN

#	Current State	Event /Condition =>Action	Next State
		=> StopTimer (CMI)	
17	RUN	CM_Write.ind () => StopTimer (CMI)	RUN
18	RUN	CM_DCControl.rsp () => StartTimer (CMI, CMInitiatorActivityTimeout)	RUN
19	RUN	CM_DCControl.ind () => StopTimer (CMI)	RUN

5.6.3.8.5 Functions, Macros, Timers and Variables

Table 923 contains the functions, macros, timers and variables used by the CMSM and their arguments and their descriptions.

Table 923 – 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 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 924.

Table 924 – 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 925.

Table 925 – 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 141.

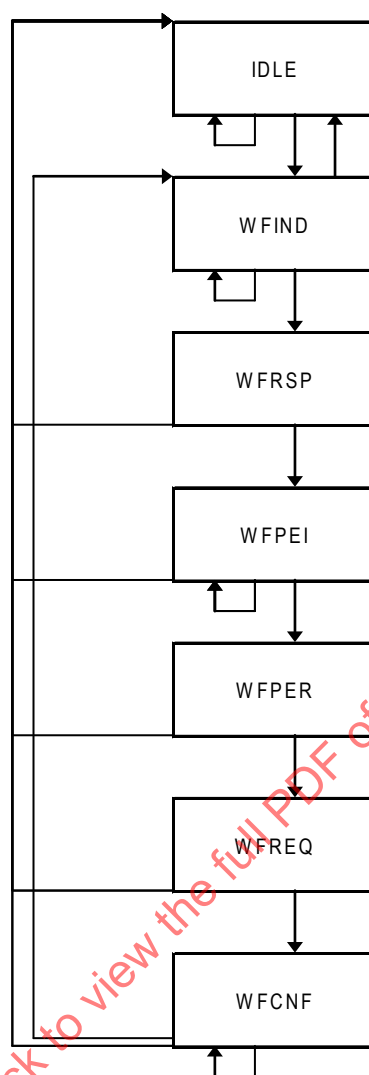


Figure 141 – State transition diagram of CMPBE

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
WPPER	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 make a consistent update of the parameters of submodules using the wellknown ApplRdy model. The IO device handles this sequence by the CMPBE.

The IO device stores all Alarms from the involved submodules during the PrmBegin PrmEnd ApplRdy sequence.

Any overlapping Alarm, except the Plug or Release, shall be proceed 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 rates all stored Alarms from the involved submodules and removes all dispensable Alarms before issuing ApplRdy.

5.6.3.9.4 CMPBE state table

Table 926 contains the complete description of the CMPBE state machine.

Table 926 – 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 ()	WPPER
10	WFPEI	RM_Dcontrol.ind () /ControlBlock != PrmEnd =>	WFPEI

#	Current State	Event /Condition =>Action	Next State
		ignore	
11	WFPEI	CMDEV_state_ind () /state == ABORT => ignore	IDLE
12	WPPER	CM_Dcontrol.rsp () /ControlBlock == PrmEnd => RM_Dcontrol.rsp (+)	WFREQ
13	WPPER	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 927 contains the functions, macros, timers and variables used by the CMPBE and their arguments and their descriptions.

Table 927 – 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 kind of Alarms for any involved submodule (list given by the PrmBegin).
AlarmProceedingBlocked	Macro	This local macro stops the proceeding of all kind 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 service primitives including their associated parameters issued or received by Discovery Multicast Communication Protocol Machine (CMDMC) are described in the service definition and shown in Table 928.

Table 928 – 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 a M-Provider station name has failed.
DCP_Identify.cnf (+)	DCPMCS	CMDMC	CREP	This service primitive indicates that the address resolution of a M-Provider station name has succeeded.
DCP_Identify.req	CMDMC	DCPMCS	CREP, DA, ListOfFilter, ResponseDelay	The service Identify starts the address resolution of the station name of a 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 929.