

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 61158-6-10:2010



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2010 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
Email: inmail@iec.ch
Web: www.iec.ch

About the IEC

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

About IEC publications

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

- Catalogue of IEC publications: www.iec.ch/searchpub

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 It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

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

- Electropedia: www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

INTERNATIONAL STANDARD



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

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

PRICE CODE **XH**

ICS 25.04.40; 35.100.70; 35.110

ISBN 978-2-88912-128-1

CONTENTS

FOREWORD.....	20
INTRODUCTION.....	22
1 Scope.....	24
1.1 General.....	24
1.2 Specifications.....	24
1.3 Conformance.....	25
2 Normative references	25
3 Terms, definitions, abbreviations, symbols and conventions	27
3.1 Referenced terms and definitions	27
3.2 Additional terms and definitions for distributed automation	28
3.3 Additional terms and definitions for decentralized periphery.....	30
3.4 Additional abbreviations and symbols for distributed automation	37
3.5 Additional abbreviations and symbols for decentralized periphery	38
3.6 Additional abbreviations and symbols for media redundancy	39
3.7 Conventions	39
3.8 Conventions used in state machines	46
4 Application layer protocol specification for common protocols.....	49
4.1 FAL syntax description	49
4.2 Transfer syntax	52
4.3 Discovery and basic configuration.....	64
4.4 Precision time control.....	96
4.5 Media redundancy.....	162
4.6 Real-time cyclic.....	187
4.7 Real-time acyclic.....	206
4.8 Remote procedure call.....	219
4.9 Link layer discovery.....	236
4.10 MAC bridges.....	244
4.11 Virtual bridges.....	259
4.12 IP suite.....	269
4.13 Domain name system.....	273
4.14 Dynamic host configuration	273
4.15 Simple network management.....	273
4.16 Common DLL Mapping Protocol Machines.....	283
5 Application layer protocol specification for distributed automation	290
5.1 FAL syntax description	290
5.2 Transfer syntax	315
5.3 FAL protocol state machines	317
5.4 AP Context State Machine.....	318
5.5 FAL Service Protocol Machines.....	319
5.6 Application Relationship Protocol Machine (ARPM)	418
5.7 DLL Mapping Protocol Machines	422
5.8 Protocol options	426
6 Application layer protocol specification for decentralized periphery.....	426
6.1 FAL syntax description	426
6.2 Transfer syntax	439
6.3 FAL protocol state machines	575

6.4	AP-Context state machine	577
6.5	FAL Service Protocol Machines	577
6.6	Application Relationship Protocol Machines	659
6.7	DLL Mapping Protocol Machines	739
Annex A (informative)	Filtering Data Base.....	740
Annex B (informative)	Establishing of a companion AR.....	743
Annex C (informative)	Establishing of a device access AR.....	744
Annex D (informative)	Establishing of an AR (simple procedure).....	745
Annex E (informative)	Establishing of an AR (accelerated procedure).....	746
Annex F (informative)	Establishing of an AR (fast startup procedure).....	749
Annex G (informative)	Example of the upload, storage, and retrieval procedure.....	750
Annex H (informative)	Establishing of an AR using RT_CLASS_3 CRs	753
Annex I (informative)	Overview of the AlarmCRs.....	754
Annex J (informative)	OSI reference model layers	756
Annex K (informative)	Overview of the IO device state machines.....	757
Annex L (informative)	Overview of the IO controller state machines.....	758
Annex M (informative)	Priority regeneration	759
Annex N (informative)	Overview of the synchronization master hierarchy.....	760
Annex O (informative)	Adapted baggy pants model of the IEEE 802.1D	762
Annex P (informative)	Optimization of bandwidth usage.....	765
Annex Q (informative)	Time constraints for bandwidth allocation.....	767
Annex R (informative)	Optimization for fast startup without autonegotiation	769
Annex S (informative)	TX-error handling.....	771
Bibliography		772
Figure 1	– Common structure of specific fields.....	43
Figure 2	– Common structure of specific fields for octet 1 (high)	44
Figure 3	– Common structure of specific fields for octet 2 (low)	44
Figure 4	– Common structure of specific fields for octet 1 (high)	45
Figure 5	– Common structure of specific fields for octet 2	45
Figure 6	– Common structure of specific fields for octet 3	45
Figure 7	– Common structure of specific fields for octet 4 (low)	46
Figure 8	– Coding of the data type BinaryDate.....	54
Figure 9	– Encoding of Time Of Day value	54
Figure 10	– Encoding of Time Difference value	55
Figure 11	– Encoding of Network Time value	55
Figure 12	– Encoding of Network Time Difference value	56
Figure 13	– Timescale correspondence between PTCP_Time and CycleCounter	102
Figure 14	– Message timestamp point.....	107
Figure 15	– Four message timestamps	108
Figure 16	– Line delay protocol with follow up.....	109
Figure 17	– Line delay protocol without follow up.....	109
Figure 18	– Line delay measurement	111
Figure 19	– Model parameter for GSDML usage	112

Figure 20 – Bridge delay measurement.....	113
Figure 21 – Delay accumulation.....	113
Figure 22 – Worst case accumulated time deviation of synchronization	114
Figure 23 – Scheme for measurement of deviation	115
Figure 24 – Measurement of deviation	115
Figure 25 – Sending Sync-Frame without Follow Up-Frame	116
Figure 26 – Sending Sync- Frame with FollowUp-Frame	116
Figure 27 – Forwarding Sync- and FollowUp-Frame.....	117
Figure 28 – Transition between Synchronization Variants	118
Figure 29 – State transition diagram of delay request	119
Figure 30 – State transition diagram of delay response.....	125
Figure 31 – Overview of PTCP.....	130
Figure 32 – State transition diagram of BMA.....	131
Figure 33 – State transition diagram of MPSM	143
Figure 34 – State transition diagram of SPSM.....	149
Figure 35 – State transition diagram of SRPM	155
Figure 36 – State transition diagram of SCHEDULER.....	160
Figure 37 – MRM protocol machine for MRP	166
Figure 38 – MRC protocol machine	173
Figure 39 – State transition diagram of RRT_RELAY.....	184
Figure 40 – CycleCounter value range	189
Figure 41 – Structure of the CycleCounter.....	190
Figure 42 – Optimized CycleCounter setting	191
Figure 43 – State transition diagram for generating events	251
Figure 44 – State transition diagram of RED RELAY	255
Figure 45 – State transition diagram of MUX.....	261
Figure 46 – State transition diagram of DEMUX	266
Figure 47 – Structuring of the protocol machines within the DMPM (bridge).....	284
Figure 48 – Error message structure.....	290
Figure 49 – Coding scheme of ITEMQUALITYDEF.....	295
Figure 50 – Relationship among protocol machines	318
Figure 51 – State transition diagram of FSPM.....	333
Figure 52 – State transition diagram of ARPM	420
Figure 53 – State transition diagram of DMPM	425
Figure 54 – FrameSendOffset vs. duration of a cycle.....	477
Figure 55 – Classification of diagnosis, maintenance and qualified	511
Figure 56 – Definition of the reserved interval.....	531
Figure 57 – Definition of PLL window	534
Figure 58 – Detection of dropped frames — appear	542
Figure 59 – Detection of dropped frames — disappear.....	542
Figure 60 – Relationship among Protocol Machines	576
Figure B.1 – Establishing of a companion AR	743
Figure C.1 – Establishing of a device access AR	744

Figure D.1 – Accelerated establishing of an IOAR (simple procedure).....	745
Figure E.1 – Accelerated establishing of an IOAR without error	747
Figure E.2 – Accelerated establishing of an IOAR with “late” error	748
Figure F.1 – Establishing of an IOAR using fast startup	749
Figure G.1 – Example of upload with storage	751
Figure G.2 – Example of retrieval with storage	752
Figure H.1 – Establishing of an AR using RT_CLASS_3 CRs	753
Figure I.1 – Overview of an AlarmCR using high priority	754
Figure I.2 – Overview of an AlarmCR using low priority	755
Figure J.1 – Assignment of the OSI reference model layers	756
Figure K.1 – Overview of the IO device state machines	757
Figure L.1 – Overview of the IO controller state machines	758
Figure N.1 – Level model for synchronization master hierarchy	760
Figure N.2 – Two level variant of the synchronization master hierarchy	761
Figure O.1 – Adapted baggy pants model of the IEEE 802.1D	762
Figure O.2 – Adapted baggy pants model of the IEEE 802.1D for a frame transmitter	763
Figure O.3 – Adapted baggy pants model of the IEEE 802.1D for a frame receiver	764
Figure P.1 – Devices build up in a linear structure	765
Figure P.2 – Propagation of frames in linear transmit direction	765
Figure P.3 – Propagation of a frames in receive direction	766
Figure Q.1 – Overview of time constraints for bandwidth allocation	767
Figure Q.2 – Calculation of the length of a period	767
Figure R.1 – Scheme of a 2-port switch	769
Figure R.2 – Scheme of 2-ports	769
Table 1 – State machine description elements	46
Table 2 – Description of state machine elements	47
Table 3 – Conventions used in state machines	47
Table 4 – IEEE 802.3 DLPDU syntax	49
Table 5 – IEEE 802.11 DLPDU syntax	50
Table 6 – IEEE 802.15.1 DLPDU syntax	50
Table 7 – SourceAddress	57
Table 8 – DCP_MulticastMACAdd	57
Table 9 – PTCP_MulticastMACAdd range 1	57
Table 10 – PTCP_MulticastMACAdd range 2	58
Table 11 – PTCP_MulticastMACAdd range 3	58
Table 12 – PTCP_MulticastMACAdd range 4	58
Table 13 – PTCP_MulticastMACAdd range 5	58
Table 14 – PTCP_MulticastMACAdd range 6	59
Table 15 – PTCP_MulticastMACAdd range 7	59
Table 16 – PTCP_MulticastMACAdd range 8	59
Table 17 – MRP Organizationally Unique Identifier (OUI)	59
Table 18 – MRPMulticastMACAdd	60

Table 19 – LT (Length/Type).....	60
Table 20 – TagControlInformation.Priority.....	61
Table 21 – FrameID range 1	61
Table 22 – FrameID range 2	61
Table 23 – FrameID range 3	62
Table 24 – FrameID range 4	62
Table 25 – FrameID range 5	62
Table 26 – FrameID range 6	62
Table 27 – FrameID range 7	62
Table 28 – FrameID range 8	63
Table 29 – FrameID range 9	63
Table 30 – FrameID range 10	63
Table 31 – FrameID range 11	63
Table 32 – FrameID range 12	64
Table 33 – FrameID range 13	64
Table 34 – DCP APDU syntax.....	64
Table 35 – DCP substitutions.....	66
Table 36 – ServiceID	68
Table 37 – ServiceType for request	68
Table 38 – ServiceType for response.....	68
Table 39 – ResponseDelayFactor	69
Table 40 – List of options.....	70
Table 41 – List of suboptions for option IPOption.....	70
Table 42 – List of suboptions for option DevicePropertiesOption.....	70
Table 43 – List of suboptions for option DHCPOption.....	70
Table 44 – List of suboptions for option ControlOption	71
Table 45 – List of suboptions for option DeviceInitiativeOption.....	71
Table 46 – List of suboptions for option AllSelectorOption	71
Table 47 – List of suboptions for option ManufacturerSpecificOption.....	71
Table 48 – SuboptionDHCP	73
Table 49 – Coding of DCPBlocklength in conjunction with SuboptionStart.....	74
Table 50 – Coding of DCPBlocklength in conjunction with SuboptionStop	74
Table 51 – Coding of DCPBlocklength in conjunction with SuboptionSignal	74
Table 52 – Coding of DCPBlocklength in conjunction with SuboptionFactoryReset	75
Table 53 – Coding of DCPBlocklength in conjunction with SuboptionDeviceInitiative	75
Table 54 – BlockQualifier with option IP.....	76
Table 55 – BlockQualifier with option DeviceProperties, DHCP, and ManufacturerSpecific	76
Table 56 – BlockError	77
Table 57 – BlockInfo for SuboptionIPParameter	77
Table 58 – Bit 1 and Bit 0 of BlockInfo for SuboptionIPParameter	77
Table 59 – Bit 7 of BlockInfo for SuboptionIPParameter	77
Table 60 – BlockInfo for all other suboptions	78

Table 61 – DeviceInitiativeValue	78
Table 62 – SignalValue	78
Table 63 – DeviceRoleDetails	80
Table 64 – IPAddress	80
Table 65 – Subnetmask	81
Table 66 – StandardGateway	82
Table 67 – DCPUCS state table	85
Table 68 – DCPUCR state table	89
Table 69 – DCPMCS state table	91
Table 70 – DCPMCR state table	93
Table 71 – DCPHMCS state table	95
Table 72 – DCPHMCR state table	96
Table 73 – PTCP APDU syntax	96
Table 74 – PTCP substitutions	97
Table 75 – PTCP_TLVHeader.Type	98
Table 76 – PTCP_Delay10ns	98
Table 77 – PTCP_Delay1ns_Byte.Value	98
Table 78 – PTCP_Delay1ns	99
Table 79 – PTCP_Delay1ns_FUP	99
Table 80 – PTCP_SequenceID	99
Table 81 – PTCP_SubType for OUI (=00-0E-CF)	100
Table 82 – PTCP_NanoSeconds	100
Table 83 – PTCP_Flags_LeapSecond	101
Table 84 – Timescale correspondence between MJD, UTC, and PTCP_EPOCHNumber	101
Table 85 – Timescale correspondence between PTCP_EPOCHNumber, PTCP_Second, PTCP_Nanosecond, CycleCounter, and SendClockFactor	101
Table 86 – PTCP_MasterPriority1.Priority for SyncID == 0 and SyncProperties.Role == 2	103
Table 87 – PTCP_MasterPriority1.Priority for SyncID == 0 and SyncProperties.Role == 1	103
Table 88 – PTCP_MasterPriority1.Priority for SyncID == 1 and SyncProperties.Role == 2	103
Table 89 – PTCP_MasterPriority1.Priority for SyncID == 1 and SyncProperties.Role == 1	103
Table 90 – PTCP_MasterPriority1.Level	103
Table 91 – PTCP_MasterPriority2	104
Table 92 – PTCP_ClockClass for SyncID == 0 (clock synchronization)	104
Table 93 – PTCP_ClockClass for SyncID == 1 (time synchronization)	105
Table 94 – PTCP_ClockAccuracy	106
Table 95 – PTCP_ClockVariance	106
Table 96 – PTCP_T2TimeStamp	107
Table 97 – DelayRequest state table	121
Table 98 – Macros used by DelayRequest	124
Table 99 – Functions used by DelayRequest	124
Table 100 – DelayResponse state table	127

Table 101 – Macros used by DelayResponse	128
Table 102 – Functions used by DelayResponse	129
Table 103 – BMA state table	133
Table 104 – BMA best remote sync master (RSM) state table	137
Table 105 – BMA get best sync master (GBSM) state table	139
Table 106 – Macros used by BMA	141
Table 107 – Functions used by BMA	142
Table 108 – MPSM state table	144
Table 109 – Macros used by MPSM	147
Table 110 – Functions used by MPSM	148
Table 111 – SPSM state table	150
Table 112 – Macros used by SPSM	153
Table 113 – Functions used by SPSM	154
Table 114 – SRPM state table	156
Table 115 – Macros used by SRPM	157
Table 116 – Functions used by the SRPM	158
Table 117 – Truth table for one SyncID	159
Table 118 – SCHEDULER state table	161
Table 119 – Functions used by the SCHEDULER	162
Table 120 – MRP APDU syntax	162
Table 121 – MRP substitutions	163
Table 122 – MRP_TLVHeader.Type	163
Table 123 – MRP_Prio	164
Table 124 – MRP_PortRole	164
Table 125 – MRP_RingState	164
Table 126 – MRP_Interval	165
Table 127 – MRP_Transition	165
Table 128 – MRP_TimeStamp	165
Table 129 – MRP_DomainUUID	165
Table 130 – Local variables of MRM protocol machine	167
Table 131 – MRM state machine	168
Table 132 – Local variables of MRC protocol machine	174
Table 133 – MRC state machine	175
Table 134 – Functions	180
Table 135 – FDB Clear Timer	182
Table 136 – Topology Change Timer	183
Table 137 – RRT_RELAY state table	185
Table 138 – Macros used by RRT_RELAY	187
Table 139 – RTC APDU syntax	187
Table 140 – RTC substitutions	188
Table 141 – CycleCounter Difference	189
Table 142 – DataStatus.State	191
Table 143 – DataStatus.Redundancy	192

Table 144 – DataStatus.DataValid	192
Table 145 – DataStatus.ProviderState	192
Table 146 – DataStatus.StationProblemIndicator	192
Table 147 – DataStatus.reserved_3	192
Table 148 – DataStatus.Ignore of a frame	193
Table 149 – TransferStatus for RT_CLASS_3	193
Table 150 – IOxS.Extension	194
Table 151 – IOCS.Instance	194
Table 152 – IOxS.DataState	194
Table 153 – PPM state table	197
Table 154 – Functions used by the PPM	200
Table 155 – CPM state table	202
Table 156 – Functions used by the CPM	206
Table 157 – RTA APDU syntax	206
Table 158 – RTA substitutions	207
Table 159 – PDUType.Type	208
Table 160 – PDUType.Version	208
Table 161 – VarPartLen	209
Table 162 – APMS state table	211
Table 163 – Functions used by the APMS and APMR	214
Table 164 – A_Timer_add	214
Table 165 – A_Timer_event	215
Table 166 – A_Timer_remove	215
Table 167 – APMR state table	217
Table 168 – RPC APDU syntax	219
Table 169 – RPC substitutions	220
Table 170 – RPCPacketType	221
Table 171 – RPCFlags	221
Table 172 – RPCFlags2	222
Table 173 – RPCDRep.Character- and IntegerEncoding	222
Table 174 – RPCDRep Octet 2 – Floating Point Representation	223
Table 175 – RPCObjectUUID.Data4	223
Table 176 – RPCObjectUUID for PNIO	224
Table 177 – RPCInterfaceUUID for PNIO	224
Table 178 – RPCInterfaceUUID for the RPC end point mapper	224
Table 179 – RPCOperationNmb (IO device, controller and supervisor)	226
Table 180 – RPCOperationNmb for endpoint mapper	226
Table 181 – RPCDataRepresentationUUID – defined values	228
Table 182 – RPCInquiryType	229
Table 183 – RPCEPMapStatus	231
Table 184 – Values of NCAFaultStatus	234
Table 185 – Values of NCAResjectStatus	235
Table 186 – LLDP APDU syntax	236

Table 187 – LLDP substitutions	237
Table 188 – LLDP_PNIO_SubType	238
Table 189 – PTCP_PortRxDelayLocal	239
Table 190 – PTCP_PortRxDelayRemote	239
Table 191 – PTCP_PortTxDelayLocal	239
Table 192 – PTCP_PortTxDelayRemote	239
Table 193 – CableDelayLocal	240
Table 194 – RTClass2_PortStatus.State	240
Table 195 – RTClass3_PortStatus.State	240
Table 196 – MRRT_PortStatus.State	241
Table 197 – LLDP_RedOrangePeriodBegin.Offset	241
Table 198 – LLDP_RedOrangePeriodBegin.Valid	241
Table 199 – LLDP_OrangePeriodBegin.Offset	242
Table 200 – LLDP_OrangePeriodBegin.Valid	242
Table 201 – LLDP_GreenPeriodBegin.Offset	242
Table 202 – LLDP_GreenPeriodBegin.Valid	243
Table 203 – LLDP_LengthOfPeriod.Length	243
Table 204 – LLDP_LengthOfPeriod.Valid	243
Table 205 – Primitives issued by LLDP to RTC3PSM	245
Table 206 – Primitives issued by IEEE 802.3 to RTC3PSM	245
Table 207 – Primitives issued by PTCP to RTC3PSM	245
Table 208 – Primitives issued by IEEE 802.1D to RTC3PSM	245
Table 209 – Primitives issued by RTC3PSM to IEEE 802.1D	245
Table 210 – RTC3PSM state table	246
Table 211 – RTC3PSM function table	250
Table 212 – Event function table	251
Table 213 – Primitives issued by LLDP to RTC2PSM	252
Table 214 – Primitives issued by IEEE 802.3 to RTC2PSM	252
Table 215 – Primitives issued by PTCP to RTC2PSM	252
Table 216 – Primitives issued by IEEE 802.1D to RTC2PSM	252
Table 217 – Primitives issued by RTC2PSM to IEEE 802.1D	252
Table 218 – RTC2PSM state table	253
Table 219 – RTC2PSM function table	255
Table 220 – RED RELAY state table	257
Table 221 – RED RELAY function table	259
Table 222 – RED RELAY macro table	259
Table 223 – Primitives issued by MUX to MAC	260
Table 224 – Primitives issued by MAC to MUX	260
Table 225 – MUX state table	262
Table 226 – MUX function table	264
Table 227 – Primitives issued by MAC to DEMUX	265
Table 228 – DEMUX state table	267
Table 229 – DEMUX function table	268

Table 230 – DEMUX macro table	269
Table 231 – IP/UDP APDU syntax	269
Table 232 – IP/UDP substitutions	270
Table 233 – UDP_SrcPort	271
Table 234 – UDP_DstPort	271
Table 235 – IP_DstIPAddress	271
Table 236 – IP Multicast DstIPAddress according to RFC 2365	272
Table 237 – Enterprise number	273
Table 238 – LMPM state table	286
Table 239 – LMPM macros table	289
Table 240 – LMPM function table	289
Table 241 – Error messages	291
Table 242 – VARTYPE values	294
Table 243 – ITEMQUALITYDEF values	296
Table 244 – STATEDEF values	299
Table 245 – GROUPEXCEPTIONDEF values	299
Table 246 – ACCESSRIGHTSDEF values	299
Table 247 – PERSISTDEF values	299
Table 248 – UUID values	303
Table 249 – Data format for serialized connection data	316
Table 250 – Calculation of the RT reference data size	317
Table 251 – Primitives issued by FAL User to FSPM	320
Table 252 – Primitives issued by FSPM to FAL User	327
Table 253 – FSPM state descriptions	333
Table 254 – FSPM state table	334
Table 255 – Primitives issued by FSPM to ARPM	419
Table 256 – Primitives issued by ARPM to FSPM	419
Table 257 – Parameters used with primitives exchanged between FSPM and ARPM	420
Table 258 – ARPM state descriptions	420
Table 259 – ARPM state table	420
Table 260 – Primitives issued by ARPM to DMPM	422
Table 261 – Primitives issued by DMPM to ARPM	423
Table 262 – Parameters used with primitives exchanged between ARPM and DMPM	423
Table 263 – Primitives issued by DMPM to ORPC model	424
Table 264 – Primitives issued by ORPC model to DMPM	424
Table 265 – Parameters used with primitives exchanged between DMPM and ORPC model	425
Table 266 – DMPM state descriptions	425
Table 267 – DMPM state table	426
Table 268 – IO APDU substitutions	428
Table 269 – BlockType	440
Table 270 – BlockVersionHigh	444
Table 271 – BlockVersionLow	444

Table 272 – AlarmType	445
Table 273 – AlarmSpecifier.ChannelDiagnosis	446
Table 274 – AlarmSpecifier.ManufacturerSpecificDiagnosis	446
Table 275 – AlarmSpecifier.SubmoduleDiagnosisState	446
Table 276 – AlarmSpecifier.ARDiagnosticsState	447
Table 277 – API	447
Table 278 – SlotNumber	447
Table 279 – SubslotNumber	448
Table 280 – Grouping of DiagnosisData	450
Table 281 – Index (user specific)	451
Table 282 – Index (subslot specific)	452
Table 283 – Index (slot specific)	453
Table 284 – Index (AR specific)	454
Table 285 – Index (API specific)	455
Table 286 – Index (device specific)	456
Table 287 – ARTYPE	457
Table 288 – IOCRMulticastMACAdd using RT_CLASS_UDP	458
Table 289 – IOCRMulticastMACAdd using RT_CLASS_2 or RT_CLASS_3	458
Table 290 – Type 10 OUI	458
Table 291 – ARProperties.State	459
Table 292 – ARProperties.SupervisorTakeoverAllowed	459
Table 293 – ARProperties.ParametrizationServer	459
Table 294 – ARProperties.DeviceAccess	459
Table 295 – ARProperties.CompanionAR	460
Table 296 – ARProperties.AcknowledgeCompanionAR	460
Table 297 – ARProperties.PullModuleAlarmAllowed	460
Table 298 – IOCRProperties.RTClass	461
Table 299 – IOCRProperties.MediaRedundancy	461
Table 300 – IOCRTagHeader.IOCRVLANID	462
Table 301 – IOCRTagHeader.IOUserPriority	462
Table 302 – IOCRType	462
Table 303 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess:=0	462
Table 304 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess:=1	463
Table 305 – LengthIOCS	463
Table 306 – LengthIOPS	464
Table 307 – AlarmCRProperties.Priority	464
Table 308 – AlarmCRProperties.Transport	464
Table 309 – AlarmCRTagHeaderHigh.AlarmCRVLANID	465
Table 310 – AlarmCRTagHeaderHigh.AlarmUserPriority	465
Table 311 – AlarmCRTagHeaderLow.AlarmCRVLANID	465
Table 312 – AlarmCRTagHeaderLow.AlarmUserPriority	465
Table 313 – AlarmSequenceNumber	466
Table 314 – AlarmCRType	466

Table 315 – RTATimeoutFactor	466
Table 316 – RTARetries.....	466
Table 317 – AddressResolutionProperties.Protocol.....	467
Table 318 – AddressResolutionProperties.Factor.....	467
Table 319 – MCITimeoutFactor	468
Table 320 – ModuleIdentNumber	468
Table 321 – SubmoduleIdentNumber	469
Table 322 – ControlBlockProperties in conjunction with ControlCommand.ApplicationReady	470
Table 323 – ControlBlockProperties in conjunction with the other values of the field ControlCommand	470
Table 324 – ControlCommand.PrmEnd	471
Table 325 – ControlCommand.ApplicationReady.....	471
Table 326 – ControlCommand.Release	471
Table 327 – ControlCommand.Done	471
Table 328 – ControlCommand.ReadyForCompanion	471
Table 329 – ControlCommand.ReadyForRT_CLASS_3	472
Table 330 – DataDescription.Type	472
Table 331 – Values of SendClockFactor	473
Table 332 – Values of ReductionRatio	474
Table 333 – Values of Phase	475
Table 334 – Values of Sequence	475
Table 335 – DataHoldFactor of a frame	476
Table 336 – WatchdogFactor of a frame	476
Table 337 – Values of FrameSendOffset.....	476
Table 338 – Values of ErrorCode for negative responses.....	478
Table 339 – Values of ErrorDecode	478
Table 340 – Coding of ErrorCode1 with ErrorDecode PNIORW	479
Table 341 – Values of ErrorCode1 and ErrorCode2 for ErrorDecode with the value PNIO	480
Table 342 – Values of ErrorCode2 for ErrorCode1 = RPC	486
Table 343 – ModuleState	486
Table 344 – SubmoduleState.AddInfo	487
Table 345 – SubmoduleState.QualifiedInfo	487
Table 346 – SubmoduleState.MaintenanceRequired	487
Table 347 – SubmoduleState.MaintenanceDemanded	487
Table 348 – SubmoduleState.DiagInfo	488
Table 349 – SubmoduleState.ARInfo	488
Table 350 – SubmoduleState.IdentInfo	488
Table 351 – SubmoduleState.FormatIndicator.....	488
Table 352 – SubmoduleState.Detail	489
Table 353 – SubmoduleProperties.Type.....	489
Table 354 – SubmoduleProperties.SharedInput	490
Table 355 – SubmoduleProperties.ReduceInputSubmoduleDataLength	490

Table 356 – SubmoduleProperties.ReduceOutputSubmoduleDataLength	490
Table 357 – SubmoduleProperties.DiscardIOXS	490
Table 358 – SubstitutionMode	491
Table 359 – SubstituteActiveFlag	491
Table 360 – InitiatorUDPRTPort	492
Table 361 – ResponderUDPRTPort	492
Table 362 – InitiatorRPCServerPort	492
Table 363 – ResponderRPCServerPort	493
Table 364 – IM_Hardware_Revision	493
Table 365 – IM_SWRevision_Functional_Enhancement	494
Table 366 – IM_SWRevision_Bug_Fix	494
Table 367 – IM_SWRevision_Internal_Change	494
Table 368 – IM_Revision_Counter	494
Table 369 – IM_Profile_ID	494
Table 370 – IM_Profile_Specific_Type	495
Table 371 – IM_Version_Major	495
Table 372 – IM_Version_Minor	495
Table 373 – IM_Date	496
Table 374 – UserStructureIdentifier	498
Table 375 – ChannelErrorType	500
Table 376 – ChannelNumber	501
Table 377 – ChannelProperties_Type	502
Table 378 – Valid combinations within ChannelProperties	503
Table 379 – Valid combinations for Alarmnotification and RecordDataRead(DiagnosisData)	504
Table 380 – ChannelProperties_Specifier	505
Table 381 – ChannelProperties.Direction	505
Table 382 – ExtChannelErrorType	505
Table 383 – ExtChannelErrorType for ChannelErrorType 0 – 0x7FFF	505
Table 384 – ExtChannelErrorType for ChannelErrorType “Data transmission impossible”	506
Table 385 – ExtChannelErrorType for ChannelErrorType “Remote mismatch”	506
Table 386 – ExtChannelErrorType for ChannelErrorType “Media redundancy mismatch”	507
Table 387 – ExtChannelErrorType for ChannelErrorType “Sync mismatch” and for ChannelErrorType “Time mismatch”	507
Table 388 – ExtChannelErrorType for ChannelErrorType “Isochronous mode mismatch”	507
Table 389 – ExtChannelErrorType for ChannelErrorType “Multicast CR mismatch”	508
Table 390 – ExtChannelErrorType for ChannelErrorType “Fiber optic mismatch”	508
Table 391 – ExtChannelErrorType for ChannelErrorType “Network component function mismatch”	508
Table 392 – Values for Accumulative Info	509
Table 393 – Values for “Fiber optic mismatch” – “Power Budget”	509
Table 394 – Values for “Network component function mismatch” – “Frame dropped”	509

Table 395 – Values for “Remote mismatch” – “Peer CableDelay mismatch”	510
Table 396 – Values for QualifiedChannelQualifier	510
Table 397 – Values for MaintenanceStatus	511
Table 398 – URRecordIndex	512
Table 399 – URRecordLength	512
Table 400 – LineDelay.Value with LineDelay.FormatIndicator == 0	513
Table 401 – LineDelay.Value with LineDelay.FormatIndicator == 1	513
Table 402 – LineDelay.FormatIndicator	513
Table 403 – RxPort	514
Table 404 – NumberOfTxPortGroups	514
Table 405 – TxPortEntry	515
Table 406 – FrameDetails.SyncFrame	516
Table 407 – FrameDetails.MeaningFrameSendOffset	516
Table 408 – MAUType	517
Table 409 – Valid combinations between MAUType and LinkState	518
Table 410 – CheckSyncMode.CableDelay	518
Table 411 – CheckSyncMode.SyncMaster	519
Table 412 – MAUTypeMode.Check	519
Table 413 – DomainBoundaryIngress	520
Table 414 – DomainBoundaryEgress	521
Table 415 – MulticastBoundary	521
Table 416 – PeerToPeerBoundary	522
Table 417 – DCPBoundary	522
Table 418 – LinkState.Link	523
Table 419 – LinkState.Port	523
Table 420 – MediaType	523
Table 421 – MaxBridgeDelay	524
Table 422 – NumberOfPorts	524
Table 423 – MaxPortTxDelay	524
Table 424 – MaxPortRxDelay	524
Table 425 – MaxLineRxDelay	525
Table 426 – YellowTime	525
Table 427 – StartOfRedFrameID	525
Table 428 – EndOfRedFrameID	526
Table 429 – Dependencies of StartOfRedFrameID and EndOfRedFrameID	526
Table 430 – NumberOfAssignments	526
Table 431 – NumberOfPhases	527
Table 432 – AssignedValueForReservedBegin	527
Table 433 – AssignedValueForOrangeBegin	527
Table 434 – AssignedValueForReservedEnd	528
Table 435 – Values of RedOrangePeriodBegin	528
Table 436 – Dependencies of RedOrangePeriodBegin, OrangePeriodBegin, and GreenPeriodBegin	528

Table 437 – Values of OrangePeriodBegin.....	529
Table 438 – Values of GreenPeriodBegin	529
Table 439 – EtherType	529
Table 440 – SyncProperties.Role	530
Table 441 – SyncProperties.SyncID	530
Table 442 – Dependencies of ReservedIntervalBegin and ReservedIntervalEnd	531
Table 443 – SyncSendFactor	531
Table 444 – PTCPTimeoutFactor	532
Table 445 – PTCPTakeoverTimeoutFactor	532
Table 446 – PTCPMasterStartupTime	533
Table 447 – PLLWindow	533
Table 448 – TimeDataCycle	534
Table 449 – TimeIOInput	534
Table 450 – TimeIOOutput	535
Table 451 – TimeIOInputValid	535
Table 452 – TimeIOOutputValid	535
Table 453 – ControllerApplicationCycleFactor	535
Table 454 – MRP_Role	536
Table 455 – MRP_RTMode.RTClass1_2	536
Table 456 – MRP_TOPchgT	536
Table 457 – MRP_TOPNRmax	537
Table 458 – MRP_TSTshortT	537
Table 459 – MRP_TSTdefaultT	537
Table 460 – MRP_TSTNRmax	538
Table 461 – MRP_LNKdownT	538
Table 462 – MRP_LNKupT	538
Table 463 – MRP_LNKNRmax	539
Table 464 – MRP_RTState	539
Table 465 – MRP_Check.MediaRedundancyManager	539
Table 466 – MRP_Check.MRP_DomainUUID	539
Table 467 – VendorBlockType	540
Table 468 – FiberOpticType	540
Table 469 – FiberOpticCableType	540
Table 470 – FiberOpticPowerBudgetType.Value	541
Table 471 – FiberOpticPowerBudgetType.CheckEnable	541
Table 472 – NCDropBudgetType.Value	541
Table 473 – NCDropBudgetType.CheckEnable	542
Table 474 – FSHelloMode.Mode	543
Table 475 – FSHelloInterval	543
Table 476 – FSHelloRetry	544
Table 477 – FSHelloDelay	544
Table 478 – FSPParameterMode.Mode	544
Table 479 – FSPParameterUUID	545

Table 480 – ArgsLength check.....	545
Table 481 – ARBlockReq – request check	547
Table 482 – IOCRBlockReq – request check.....	548
Table 483 – AlarmCRBlockReq – request check	552
Table 484 – ExpectedSubmoduleBlockReq – request check	553
Table 485 – PrmServerBlock – request check.....	554
Table 486 – MCRBlockReq – request check	555
Table 487 – ARRPCBlockReq – request check	555
Table 488 – ArgsLength check.....	555
Table 489 – ARBlockRes – response check	556
Table 490 – IOCRBlockRes – response check	557
Table 491 – AlarmCRBlockRes – response check.....	558
Table 492 – ModuleDiffBlock – response check	559
Table 493 – ArgsLength check.....	560
Table 494 – ControlBlockConnect – request check	560
Table 495 – ControlBlockPlug – request check	561
Table 496 – ArgsLength check.....	561
Table 497 – ControlBlockConnect – response check.....	562
Table 498 – ControlBlockPlug – response check.....	563
Table 499 – ArgsLength check.....	563
Table 500 – ControlBlockConnect – request check	564
Table 501 – ControlBlockPlug – request check	564
Table 502 – ArgsLength check.....	564
Table 503 – ControlBlockConnect – response check.....	565
Table 504 – ControlBlockPlug – response check.....	566
Table 505 – ArgsLength check.....	566
Table 506 – ReleaseBlock – request check.....	567
Table 507 – ArgsLength check.....	567
Table 508 – ReleaseBlock – response check	568
Table 509 – ArgsLength check.....	568
Table 510 – IODWriteReqHeader – request check	569
Table 511 – ArgsLength check.....	569
Table 512 – IODWriteResHeader – response check.....	570
Table 513 – ArgsLength check.....	570
Table 514 – ArgsLength check.....	571
Table 515 – ArgsLength check.....	572
Table 516 – IODReadReqHeader – request check	573
Table 517 – RecordDataReadQuery – request check	573
Table 518 – ArgsLength check.....	574
Table 519 – IODReadResHeader – response check.....	575
Table 520 – Primitives issued by AP-Context (FAL user) to FSPMDEV	579
Table 521 – Primitives issued by FSPMDEV to AP-Context (FAL user)	589
Table 522 – FSPMDEV protocol machine for multicast communication	600

Table 523 – Functions used by AP-Context (FAL user) to FSPMDEV	609
Table 524 – Function used by FSPMDEV to AP-Context (FAL user)	613
Table 525 – Primitives issued by AP-Context (FAL user) to FSPMCTL	622
Table 526 – Primitives issued by FSPMCTL to AP-Context (FAL user)	628
Table 527 – Function used by AP-Context (FAL user) to FSPMCTL	638
Table 528 – Functions used by FSPMCTL to AP-Context (FAL user)	650
Table 529 – Primitives issued by FSPMDEV or FSPMCTL to ALPMI	659
Table 530 – Primitives issued by ALPMI to FSPMDEV or FSPMCTL	660
Table 531 – Primitives issued by CMDEV or CMCTL to ALPMI	660
Table 532 – Primitives issued by ALPMI to CMCTL or CMDEV	660
Table 533 – Primitives issued by APMR to ALPMI	661
Table 534 – Primitives issued by ALPMI to APMR	661
Table 535 – Primitives issued by APMS to ALPMI	662
Table 536 – Primitives issued by ALPMI to APMS	662
Table 537 – ALPMI state table	663
Table 538 – Primitives issued by FSPMDEV or FSPMCTL to ALPMR	666
Table 539 – Primitives issued by ALPMR to FSPMDEV or FSPMCTL	667
Table 540 – Primitives issued by CMDEV or CMCTL to ALPMR	667
Table 541 – Primitives issued by ALPMR to CMCTL or CMDEV	667
Table 542 – Primitives issued by APMR to ALPMR	668
Table 543 – Primitives issued by ALPMR to APMR	668
Table 544 – Primitives issued by APMS to ALPMR	669
Table 545 – Primitives issued by ALPMR to APMS	669
Table 546 – ALPMR state table	670
Table 547 – Primitives issued by CMCTL to NRPM	675
Table 548 – Primitives issued by NRPM to CMCTL	676
Table 549 – Primitives issued by other machines to NRPM	678
Table 550 – Primitives issued by NRPM to other machines	679
Table 551 – NRPM state table	680
Table 552 – Functions used by the NRPM and RMPM	685
Table 553 – Primitives issued by CMDEV to RMPM	686
Table 554 – Primitives issued by RMPM to CMDEV	687
Table 555 – Primitives issued by RPC to RMPM	687
Table 556 – Primitives issued by RMPM to RPC	688
Table 557 – Primitives issued by other machines to RMPM	689
Table 558 – Primitives issued by RMPM to other machines	690
Table 559 – RMPM state table	691
Table 560 – Macros used by RMPM	699
Table 561 – Primitives issued by FSPMDEV to CMDEV	700
Table 562 – Primitives issued by CMDEV to FSPMDEV	701
Table 563 – CMDEV state table	702
Table 564 – Macros used by CMDEV	714
Table 565 – Variables used by CMDEV	714

Table 566 – CMDEV(DA) state table	715
Table 567 – Functions used by CMDEV(DA)	717
Table 568 – Macros used by CMDEV(DA)	717
Table 569 – Primitives issued by CMDEV to NRMC	717
Table 570 – Primitives issued by NRMC to CMDEV	718
Table 571 – Primitives issued by CPM to NRMC	718
Table 572 – Primitives issued by NRMC to CPM	718
Table 573 – Primitives issued by other machines to NRMC	719
Table 574 – Primitives issued by NRMC to other machines	719
Table 575 – NRMC state table	721
Table 576 – Primitives issued by FSPMCTL to CMCTL	726
Table 577 – Primitives issued by CMCTL to FSPMCTL	727
Table 578 – CMCTL state table	729
Table A.1 – Unicast FDB entries	740
Table A.2 – Multicast FDB entries	741
Table A.3 – Broadcast FDB entry	742
Table M.1 – Priority regeneration and queue usage	759
Table Q.1 – Parameter for the equation	768
Table R.1 – Truth table	770
Table S.1 – TX-error	771

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.

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 second edition cancels and replaces the first edition published in 2007. This edition constitutes a technical revision.

The main changes with respect to the previous edition are listed below:

- corrections;
- improvements;
- optimization of the synchronization;
- optimization of the startup time from power down.

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

FDIS	Report on voting
65C/607/FDIS	65C/621/RVD

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

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

A list of all the 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.

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

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

INTRODUCTION

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

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

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

This standard is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this standard together with other standards positioned within the OSI or fieldbus reference models, otherwise incompatible systems may work together in any combination.

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

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of patents concerning Type 10 elements and possibly other types given in the normative elements of this standard.

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

WO publication	Title (WO)
WO 99/046908	Local network, especially ethernet network, with redundancy properties and redundancy manager

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

WO publication	Title (WO)
WO 99/046908	Local network, especially ethernet network, with redundancy properties and redundancy manager
WO 00/026731	Automation system and method for accessing the functionality of hardware components
WO 02/043336	System and method for the parallel transmission of real-time critical and non real-time critical data via switched data networks especially ethernet
WO 02/076033	Synchronous, clocked communication system with local input/output components and method for integrating local input/output components into such a system
WO 03/028258	Method for synchronising nodes of a communication system
WO 03/028259	Communications system and method for synchronising a communications cycle
WO 04/030284	Method for permanent redundant transmission of data telegrams in communication systems

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:

[HI]: Hirschmann Automation and Control GmbH
Stuttgarter Straße 45-51
D-72654 Neckartenzlingen
Germany

[SI]: Siemens AG
CT IP L&T
Hr. Hans-Jörg Müller
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.

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 standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 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 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; and.

The purpose of this standard 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 standard specifies the protocol of the Type 10 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) and the OSI Application Layer Structure (ISO/IEC 9545).

1.2 Specifications

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

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

2 Normative references

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

IEC 60559, *Binary floating-point arithmetic for microprocessor systems*

IEC 61158-5-10:2010¹, *Industrial communication networks – Fieldbus specifications – Application layer service definition – Type 10 elements*

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

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

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

IEEE 802-2001, *IEEE Standard for Local and metropolitan area networks: Overview and Architecture*, available at <<http://www.ieee.org>>

IEEE 802.1AB-2005, *IEEE Standard for Local and metropolitan area networks: Station and Media Access Control Connectivity Discovery*, available at <<http://www.ieee.org>>

IEEE 802.1D-2004, *IEEE Standard for Local and metropolitan area networks – Media access control (MAC) Bridges*, available at <<http://www.ieee.org>>

IEEE 802.1Q-2005, *IEEE Standard for Local and metropolitan area networks – Virtual Bridged Local Area Networks*, available at <<http://www.ieee.org>>

¹ To be published.

IEEE 802.3-2005, *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and Physical Layer specifications*, available at <<http://www.ieee.org>>

IEEE 802.11-1999, *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>>

IEEE 802.15.1-2005, *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>>

IETF RFC 768, *User Datagram Protocol*; available at <<http://www.ietf.org>>

IETF RFC 791, *Internet Protocol*; available at <<http://www.ietf.org>>

IETF RFC 792, *Internet Control Message Protocol*; available at <<http://www.ietf.org>>

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>>

IETF RFC 1034, *Domain names – concepts and facilities*; available at <<http://www.ietf.org>>

IETF RFC 1112, *Host Extensions for IP Multicasting*; available at <<http://www.ietf.org>>

IETF RFC 1518, *An Architecture for IP Address Allocation with CIDR*, available at <<http://www.ietf.org>>

IETF RFC 1519, *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*, available at <<http://www.ietf.org>>

IETF RFC 2131, *Dynamic Host Configuration Protocol*; available at <<http://www.ietf.org>>

IETF RFC 2132, *DHCP Options and BOOTP Vendor Extensions*; available at <<http://www.ietf.org>>

IETF RFC 2365, *Administratively Scoped IP Multicast*; available at <<http://www.ietf.org>>

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

IETF RFC 2674, *Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions*, available at <<http://www.ietf.org>>

IETF RFC 2737, *Entity MIB (Version 2)*, available at <<http://www.ietf.org>>

IETF RFC 2863, *The Interfaces Group MIB*, available at <<http://www.ietf.org>>

IETF RFC 3330, *Special-Use IPv4 Addresses*, available at <<http://www.ietf.org>>

IETF RFC 3418, *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)*, available at <<http://www.ietf.org>>

IETF RFC 3490, *Internationalizing Domain Names in Applications (IDNA)*; available at <<http://www.ietf.org>>

IETF RFC 3621, *Power Ethernet MIB*, available at <<http://www.ietf.org>>

IETF RFC 4836, *Definitions of Managed Objects for IEEE 802.3 Medium Attachment Units (MAUs)*, available at <<http://www.ietf.org>>

The Open Group — Publication C706, *Technical standard DCE1.1: Remote Procedure Call*, available at <<http://www.opengroup.org/onlinepubs/9629399/toc.htm>>

3 Terms, definitions, abbreviations, symbols and conventions

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 Additional terms and definitions for distributed automation

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

3.2.1

active connection control object

instance of a certain FAL class that abstracts the interconnection facility (as Consumer and Provider) of an automation device

3.2.2

configuration data base

interconnection information maintained by the ACCO ASE

3.2.3

connection

the logical link between sink and source of attributes and services at different custom interfaces of Custom RT-Auto objects

3.2.4

connection channel

description of a connection between a sink and a source of data items

3.2.5

consumer

node or sink that is receiving data from a producer

3.2.6

consumerID

unambiguous identifier within the scope of the ACCO assigned by the consumer to recognize the internal data of a configured interconnection sink.

3.2.7

data marshaling

the encoding of parameters of the FAL service primitives with respect to their interface definition

NOTE This is part of the abstract ORPC model.

3.2.8

engineering

abstract term that characterizes the client application or device responsible for configuring an automation system via interconnecting data items

3.2.9

event

an instance of a change of conditions

3.2.10

interface

collection of FAL class attributes and services that represents a specific view on the FAL class

3.2.11**interface definition language**

syntax and semantic of describing service parameters in a formal way

NOTE This description is the input for the ORPC model, especially for the ORPC wire protocol.

3.2.12**interface pointer**

key attribute that unambiguously addresses an object interface instance

3.2.13**logical device**

a certain FAL class that abstracts a software component or a firmware component as an autonomous self-contained facility of an automation device

3.2.14**method**

<object> a synonym for an operational service which is provided by the server ASE and invoked by a client

3.2.15**object remote procedure call**

model for object oriented or component based remote method invocation

3.2.16**physical device**

a certain FAL class that abstracts the hardware facilities of an automation device

3.2.17**property**

a synonym for ASE attributes which are readable or writeable via operational ASE services

NOTE These services are generally named "get_<Attribute Name>" or "set_<Attribute Name>" and correspond with the IDL keywords "propget" and "propset".

3.2.18**provider**

source of a data connection.

3.2.19**providerID**

an unambiguous identifier within the scope of the ACCO assigned by the provider to recognize the internal data of a configured connection source

3.2.20**quality code**

additional status information of a data item

3.2.21**quality code aware**

attribute of the RT-Auto class that indicates that an RT-Auto object uses a status code for its data items

3.2.22**quality code unaware**

opposite of quality code aware

3.2.23

RT-Auto

an FAL class that abstracts the automation function as a process-related component of an automation device

3.2.24

runtime object model

objects that exist in a device together with their interfaces and methods that are accessible

3.3 Additional terms and definitions for decentralized periphery

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

3.3.1

alarm

activation of an event that shows a critical state

3.3.2

alarm ack

acknowledgment of an event that shows a critical state

3.3.3

alarm data object

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

3.3.4

allocate

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

3.3.5

application

function or data structure for which data is consumed or produced

3.3.6

application layer interoperability

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

3.3.7

application objects

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

3.3.8

application process

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

3.3.9

application process identifier

distinguishes multiple application processes used in a device

3.3.10

application process object

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

NOTE 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.

3.3.11

application process object class

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

3.3.12

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

3.3.13

application relationship application service element

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

3.3.14

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 Each application process involved in the application relationship maintains its own application relationship endpoint.

3.3.15

attribute

description of an externally visible characteristic or feature of an object

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

3.3.16

backup

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

3.3.17

behavior

indication of how an object responds to particular events

3.3.18

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 The channel typically represents a single connector or clamp as a real interface of a module or sub-module. This reference is used to identify points of failure within diagnosis PDUs.

3.3.19

channel related diagnosis

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

EXAMPLE open loop

3.3.20

class

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

NOTE 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.

3.3.21

class attributes

attribute that is shared by all objects within the same class

3.3.22

class code

unique identifier assigned to each object class

3.3.23

class specific service

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

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

3.3.24

clear

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

3.3.25

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

3.3.26

common profile

a collection of device independent information and functionality providing consistency between all devices

3.3.27

communication data object

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

3.3.28

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 start-up phase

3.3.29

configuration fault

an unacceptable difference between the expected IO-Data object structuring and the real IO-Data object structuring, as detected by the server

3.3.30

configuration identifier

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

3.3.31

consume

act of receiving data from a provider

3.3.32**consumer**

node or sink receiving data from a provider

3.3.33**context management**

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

NOTE Managing includes functions such as controlling, monitoring, and diagnosing.

3.3.34**conveyance path**

unidirectional flow of APDUs across an application relationship

3.3.35**cyclic**

repetitive in a regular manner

3.3.36**data consistency**

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

3.3.37**device**

physical hardware connected to the link

NOTE A device may contain more than one node.

3.3.38**device ID**

a vendor assigned device type identification

3.3.39**device profile**

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

3.3.40**diagnosis data object**

object(s) which contains diagnosis information referenced by device/slot/subslot/index

3.3.41**diagnosis information**

all data available at the server for maintenance purposes

3.3.42**dynamic reconfiguration**

change of IO data objects without interruption of an established application relationship and continuous updating of non-changed IO data objects

3.3.43**endpoint**

one of the communicating entities involved in a connection

3.3.44

engineering

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

3.3.45

error

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

3.3.46

error class

general grouping for related error definitions and corresponding error codes

3.3.47

error code

identification of a specific type of error within an error class

3.3.48

event

instance of a change of conditions

3.3.49

extended channel related diagnosis

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

EXAMPLE Link Fail

3.3.50

frame

unit of data transfer on a link

3.3.51

identification data object

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

3.3.52

implicit AR endpoint

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

3.3.53

index

address of a record data object within an application process

3.3.54

instance

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

3.3.55

instance attributes

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

3.3.56

instantiated

object that has been created in a device

3.3.57**invocation**

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

NOTE 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.

3.3.58**IO controller**

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

NOTE This is usually a programmable controller or a distributed control system.

3.3.59**IO data object**

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

3.3.60**IO device**

field device which acts as server for IO operation

3.3.61**IO parameter server**

server for application parameter of IO devices (client)

NOTE This is usually a device to backup parameter data and to log online changes of device parameter.

3.3.62**IO subsystem**

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

3.3.63**IO supervisor**

engineering device which manages commissioning and diagnosis of an IO system

3.3.64**IO system**

system composed of all its IO subsystems

NOTE As an example a PLC with more than one IO controller (network interface) controls one IO system composed of an IO subsystems for each IO controller.

3.3.65**Isochronous mode**

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

3.3.66**member**

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

3.3.67**message**

synonym for frame

3.3.68**method**

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

3.3.69

module

hardware or logical component of a physical device

3.3.70

network

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

3.3.71

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

3.3.72

object specific service

service unique to the object class which defines it

3.3.73

operate

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

3.3.74

packet

frame

3.3.75

peer

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

3.3.76

physical device

automation or other network device

3.3.77

point-to-point connection

connection that exists between exactly two application objects

3.3.78

primary

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

NOTE Besides a primary IO AR a backup IO AR may exist. In example used for redundancy and dynamic reconfiguration of IO data.

3.3.79

provider

node or source sending data to one or many consumer

3.3.80

PTCP subdomain

certain amount of DTEs with synchronized clocks

3.3.81

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/slot/subslot/index

3.3.82**resource**

processing or information capability

3.3.83**run**

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

3.3.84**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

3.3.85**service**

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

3.3.86**slot**

address of a structural unit within an IO device

NOTE Within a modular device, a slot typically addresses a physical module. Within compact devices, a slot typically addresses a logical function or virtual module.

3.3.87**stop**

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

3.3.88**submodule**

hardware or logical component of a module

3.3.89**subslot**

address of a structural unit within a slot

NOTE 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.

3.3.90**vendor ID**

central administrative number used as manufacturer identification

3.4 Additional abbreviations and symbols for distributed automation

ACCO	Active Connection Control Object
IDL	Interface Definition Language
IP	Internet Protocol
BNF	Backus-Naur Form
DNS	Domain Name Service
LDev	Logical Device
ORPC	Object Remote Procedure Call
PDev	Physical Device
QoS	Quality of Service
QC	Quality Code
RT	Runtime
RT-Auto	Runtime Automation Object
TCP	Transmission Control Protocol

3.5 Additional abbreviations 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
APO	Application Object
AR	Application Relationship
AREP	Application Relationship End Point
ARP	Address Resolution Protocol
ARPM	Application Relationship Protocol Machine
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
BMC	Best Master Clock
CM	Context Management
CMCTL	Context Management of a IO controller
CMDEV	Context Management of a IO device
Cnf	Confirmation
CR	Communication Relationship
CREP	Communication Relationship End Point
DCE	OSF Distributed Computing Environment
DCP	Discovery and basic Configuration Protocol
DCPMCR	DCP Multicast Receiver
DCPMCS	DCP Multicast Sender
DCPUCR	DCP Unicast Receiver
DCPUCS	DCP Unicast Sender
DHCP	Dynamic Host Configuration Protocol
DIM	Device Interface Module
DL-	(as a prefix) Data Link-
DLC	Data Link Connection
DLL	Data Link Layer
DLPDU	Data Link-Protocol Data Unit
DLSDU	DL-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
FSPM	FAL Service Protocol Machines
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
Ind	Indication
IOCS	Input Output Object Consumer Status
IOPS	Input Output Object Provider Status
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

lsb	Least Significant Bit
LT	Length/Type
MAC	Medium Access Control
msb	Most Significant Bit
NCA	Network Computing Architecture
OSI	Open Systems Interconnect
PDU	Protocol Data Unit
PL	Physical Layer
PTCP	Precision Transparent Clock Protocol
QoS	Quality of Service
RED RELAY	RT_CLASS_3 Forwarding Protocol Machine
Req	Request
RMPM	Resource management protocol machine
RPC	Remote Procedure Call
Rsp	Response
RT	Real Time Protocol
RTA	Real Time Protocol Acyclic
RTC	Real Time Protocol Cyclic
RTE	Real Time Ethernet
SDU	Service Data Unit
TLV	Type Length Value (coding rule)
UDP	User Datagram Protocol
UUID	Universal Unique Identifier
VLAN	Virtual Local Area Network

3.6 Additional abbreviations and symbols for media redundancy

MRC	Media Redundancy Client
MRM	Media Redundancy Manager
MRP	Media Redundancy Protocol

3.7 Conventions

3.7.1 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 standard.

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 standard. The service specification defines the services that are provided by the ASE.

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

3.7.2 Conventions for distributed automation

3.7.2.1 IDL – Interface Definition Language

3.7.2.1.1 Basic information

The Interface Definition Language (IDL) is a language for specifying interfaces, operations (procedures or functions), parameters to these operations, and data types.

Notation

The IDL is a case-sensitive language. The syntax of IDL is described using an extended BNF (Backus-Naur Form) notation. The meaning of the BNF notation is as follows:

Brackets ([]) enclose an optional part of the syntax.

Ellipsis points (...) indicate that the left clause can be repeated either zero or more times if it is optional or one or more times if it is required. The vertical bar (|) indicates alternative productions; it is read as "or."

Language punctuation that does not conflict with punctuation characters used in the BNF notation appears in a production in the appropriate position. Language punctuation that does conflict with punctuation characters used in the BNF notation is enclosed in less-than and greater-than symbols; for example, <[>. Note particularly that when ' (single quotation) or " (double quotation) appear in a production, they are a part of the language and shall appear in IDL source.

Elements in the grammar that are capitalized are terminals of the grammar. For example, <Identifier> is not further expanded. Also, keywords of the language are terminals of the grammar. For example, the keyword boolean is not further expanded.

3.7.2.1.2 Keywords and reserved words

Some keywords are reserved words, and shall not be used as identifiers. Keywords that are not reserved may be used as identifiers, except when used as attributes (that is, within [(brackets)). Reserved words are keywords of the language that shall not be used as user identifiers.

IDL reserved words are given in the following list:

boolean	byte	case	char
const	default	double	enum
FALSE	float	handle_t	hyper
import	int	interface	long
NULL	pipe	short	small
struct	switch	TRUE	typedef
union	unsigned	void	

The following list gives the IDL keywords that are reserved when in the context of an attribute: that is, between [(brackets):

align	broadcast	broadcast	comm_status
context_handle	endpoint	first_is	handle
idempotent	ignore	implicit_handle	in
last_is	length_is	local	max_is
maybe	out	ptr	ref
size_is	string	switch_is	switch_type
transmit_as	uuid	version	

3.7.2.1.3 Identifiers

Each object is named with an unique identifier. The maximum length of an identifier is 31 characters.

Some identifiers are used as a base from which the compiler constructs other identifiers. These identifiers have further restrictions on their length. The character set for identifiers contains the alphabetic characters A to Z and a to z, the digits 0 to 9, and the _ (underbar) character. An identifier shall start with an alphabetic character or the _ (underbar) character.

3.7.2.1.4 IDL punctuation

The punctuation used in IDL consists of the following characters:

The . (dot)
 The , (comma)
 The pair ()(parentheses)
 The pair [] (brackets)
 The pair {} (braces)
 The ; (semicolon)
 The : (colon)
 The * (asterisk)
 The ' (single quote)
 The " (double quote)
 The = (equal sign)

3.7.2.1.5 Interface definition structure

An interface definition written in IDL has the following structure:

```
<interface> ::= <interface_header> { <interface_body> }
```

3.7.2.1.6 Interface header

The structure of the interface header is as follows:

```
<interface_header> ::= [<interface_attributes>] > interface <Identifier>
```

where:

```
<interface_attributes> ::= <interface_attribute> [ , <interface_attribute> ] ...
<interface_attribute> ::= uuid (<Uuid_rep>)
    | version (<Integer_literal> [ . <Integer_literal> ])
    | endpoint (<port_spec> [ , <port_spec> ] ...)
    | local
    | pointer_default (<ptr_attr>)
<port_spec> ::= <Family_string> : [<Port_string> < >]
```

If an interface defines any operations, exactly one of the uuid attribute or the local attribute shall be specified. Whichever is specified shall appear exactly once.

It is permissible to have neither the uuid nor the local attribute if the interface defines no operations. The version attribute may occur at most once.

3.7.2.1.7 Interface body

The structure of the interface body is as follows:

```
<interface_body> ::= [<import> ...] <interface_component>
[ <interface_component> ... ]
```

where:

```
<import> ::= import <import_list>;
<interface_component> ::= <export>
    | <op_declarator>;
<export> ::= <type_declarator>;
    | <const_declarator>;
    | <tagged_declarator>;
```


3.7.2.1.8 Further possibilities

It is possible to use comments. The IDL allows to declare constants. Type declarations are subdivided into base types and the fundamental integer, char, boolean, byte, void, handle_t types, constructed types, structures, unions, enumerated types, pipes and arrays.

3.7.3 Conventions for decentralized periphery

3.7.3.1 Abstract syntax conventions

3.7.3.1.1 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 “()”

If octets or groups of octets are grouped within “[]” the group can be omitted

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

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

NOTE 1 The formal PDU example

AP_PDU = 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.

3.7.3.2 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.

3.7.3.3 Conventions for the common codings of specific field 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 Figure 1.

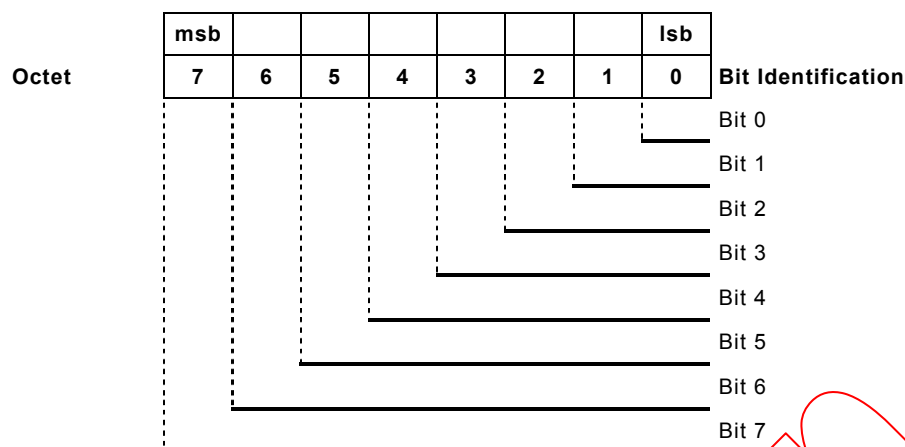


Figure 1 – Common structure of specific fields

Several bit may be grouped as group of bit. Each bit or group of bits shall be addressed by its Bit Identification (e.g. Bit 0, Bit 1 to 4). The position within the octet shall be according to the figure above. 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 8 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 msb concerning the grouped bit.

EXAMPLE 1 Description and relation for the specific field octet

Bit 0: reserved.

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

Bit 4-7: 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 1-3 equals the decimal value 2.

3.7.3.4 Conventions for the common codings of specific field 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 Figure 2 and Figure 3.

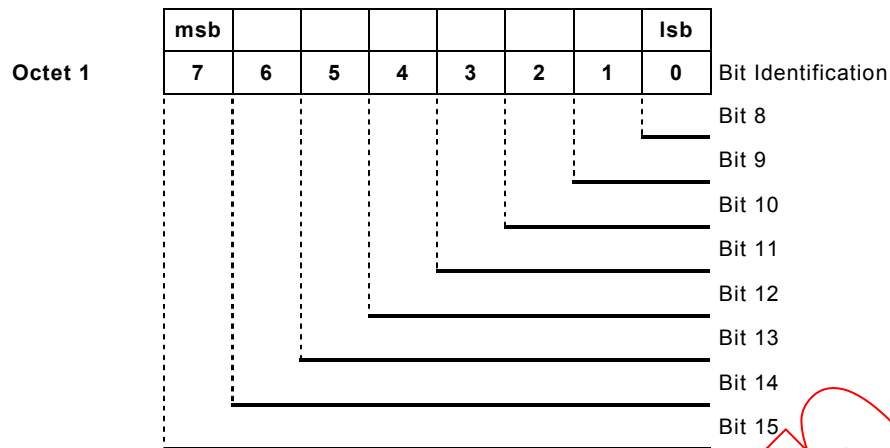


Figure 2 – Common structure of specific fields for octet 1 (high)

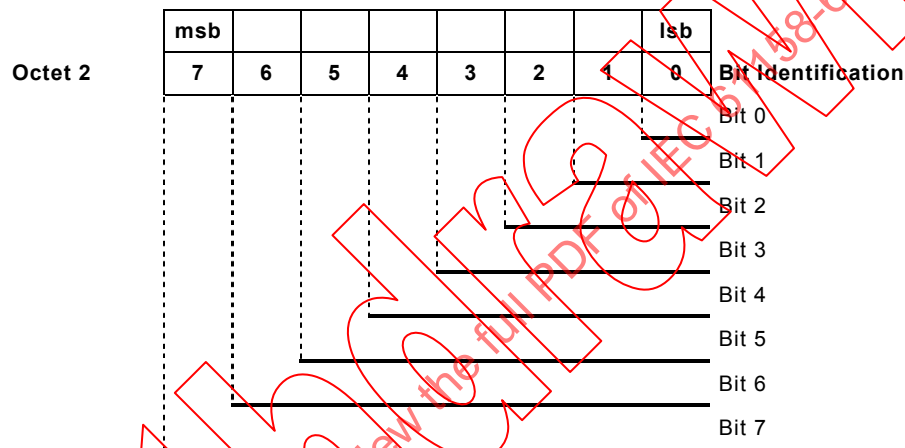


Figure 3 – Common structure of specific fields for octet 2 (low)

Several bits may be grouped as group of bit. Each bit or group of bits shall be addressed by its Bit Identification (e.g. Bit 0, Bit 1 to 4). The position within the octet shall be according to the figure above. 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 16 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 msb concerning the grouped bit.

3.7.3.5 Conventions for the common coding of specific field 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 Figure 4, Figure 5, Figure 6, and Figure 7.

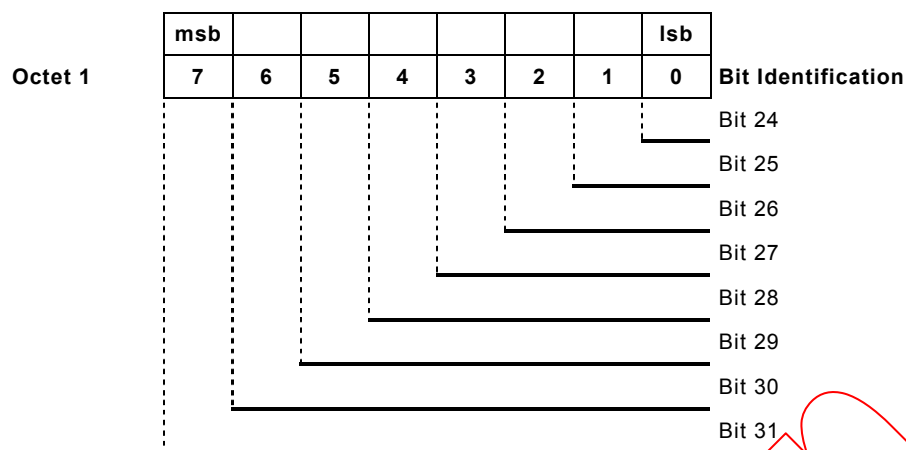


Figure 4 – Common structure of specific fields for octet 1 (high)

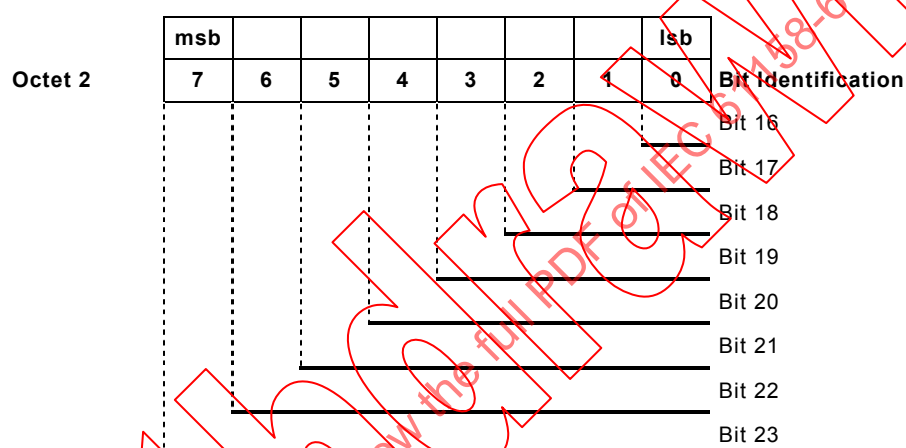


Figure 5 – Common structure of specific fields for octet 2

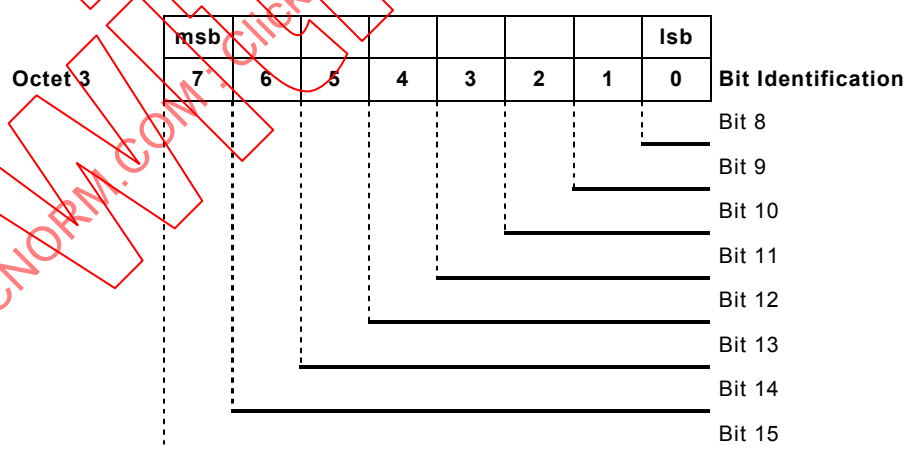


Figure 6 – Common structure of specific fields for octet 3

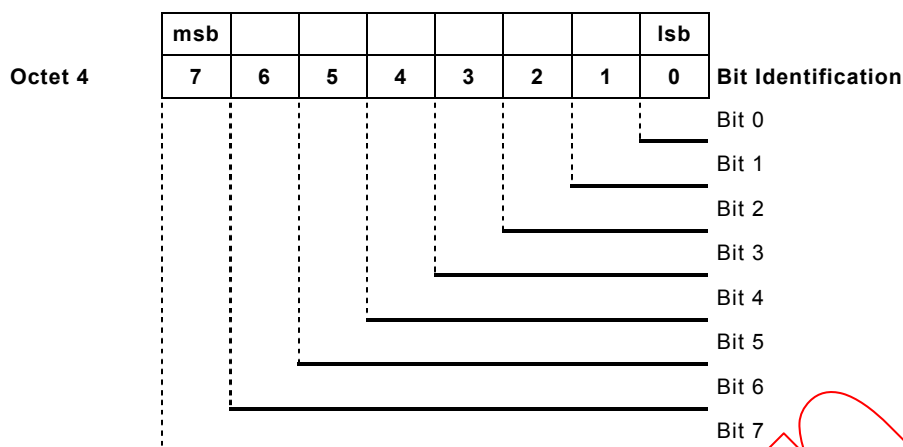


Figure 7 – Common structure of specific fields for octet 4 (low)

Bits may be grouped as group of bits. Each bit or group of bits shall be addressed by its Bit Identification (e.g. Bit 0, Bit 1 to 4). The position within the octet shall be according to the figure above. 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 32 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 msb concerning the grouped bits.

3.8 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 textual listing of the state transitions is structured as follows, see also Table 1.

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, e.g. the actions are executed and the next state is entered.

The layout of a Machine description is shown in Table 1. The meaning of the elements of a State Machine Description are shown in Table 2.

Table 1 – State machine description elements

#	Current state	Event or condition => action	Next state

Table 2 – 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 3.

Table 3 – 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.
<>	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"
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.

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”.

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.

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 4, Table 5, and Table 6 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 4 shall be according to IEEE 802.3 for the DLPDU.

Table 4 – 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 ^ MRP-PDU ^f ^ CL-RPC-PDU ^ LLDP-PDU ^e ^ ICMP-PDU
FIDAPDU	FrameID, RTC-PDU ^ RTA-PDU ^ DCP-PDU ^ PTCB-PDU ^e
VLAN	LT(=0x8100), TagControlInformation
NOTE 1 According to IEEE 802.3 the DLPDUs have a minimum length of 64 octets (excluded Preamble, Start Frame Delimiter).	
NOTE 2 For the IEEE 802.3 frames with VLAN tag is the minimum frame size increased to 68 octets in order to guaranty the minimum frame size of 64 octets after removing the VLAN tag by a bridge.	
^a The field contains at least 7 octets. ^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. The value shall be set to zero. ^d The VLAN field can be omitted in case of optimized RT transportation. The field VLAN may be set by the encoder but it may be discarded by intermediate bridges. The decoder shall accept DLPDUs with or without VLAN fields. The VLAN field should be omitted for RT_CLASS_3 PDUs conveyed in the RED PERIOD. ^e The VLAN field shall be omitted. ^f The VLAN field should be omitted by the encoder.	

4.1.1.3 IEEE 802.11

The encoding and decoding of the fields in Table 5 shall be according to IEEE 802.11 for the DLPDU.

Table 5 – 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] ^b , DLSDU, DLPDU_Padding* ^a , FrameCheckSequence
DLPDU_4	FrameControl, ReceiverAddress, TransmitterAddress, DestinationAddress, SequenceControl, SourceAddress, [QoSControl] ^b , DLSDU, DLPDU_Padding* ^a , FrameCheckSequence
DLSDU	LT, SAPDU ^ FIDAPDU
SAPDU	UDP-RTC-PDU ^ UDP-RTA-PDU ^ MRP-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	
^a The number of padding octets shall be in the range of 0..46 depending on the DLSDU size. The value shall be set to zero.	
^b This field should exist for UDP-RTC-PDU and UDP-RTA-PDU and shall exist for RTC-PDU and RTA-PDU.	

4.1.1.4 IEEE 802.15.1

The encoding and decoding of the fields in Table 6 shall be according to IEEE 802.15.1 for the DLPDU.

Table 6 – 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 ^ FIDAPDU
SAPDU	UDP-RTC-PDU ^ UDP-RTA-PDU ^ MRP-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 802.15.1-2005, 8.6.3	
NOTE 2 For definition of Packet Header see IEEE 802.15.1-2005, 8.6.4	
NOTE 3 For definition of Payload Header see IEEE 802.15.1-2005, 8.6.6.2	
NOTE 4 For definition of L2CAP Header see IEEE 802.15.1-2005, 14.3.1	
NOTE 5 For definition of BNEPType see IEEE 802.15.1-2005	
^a The minimum DLSDU size is 2 octets.	
^b This field should exist for UDP-RTC-PDU and UDP-RTA-PDU and shall exist for RTC-PDU and RTA-PDU.	
^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  $-2^7 \leq l \leq 2^7-1$ 
```

```
Integer16 ::= INTEGER (-32 768..+32 767)
```

```
-- range  $-2^{15} \leq i \leq 2^{15}-1$ 
```

```
Integer32 ::= INTEGER
```

```
-- range -231 <= i <= 231-1
```

Integer64 ::= INTEGER

```
-- range -263 ≤ I ≤ 263-1
```

4.1.2.3 Notation for the Unsigned type

```
Unsigned8 ::= INTEGER (0..255)
```

```
-- range 0 <= i <= 28-1
```

Unsigned16 ::= INTEGER (0..65 535)

range $0 \leq i \leq 2^{16}-1$

Unsigned32 ::= INTEGER

```
-- range 0 <= i <= 232-1
```

Unsigned64 ::= INTEGER

--range 0 <= i <= 2⁶⁴-1

4.1.2.4 Notation for the Floating Point type

Floating32 ::= BIT_STRING SIZE (4)

- IEC 60559 Single precision

~~Floating64 ::= BIT STRING SIZE (8)~~

```
-- IEC 60559 Double precision
```

4.1.2.5 Notation for the OctetString type

~~OctetString ::= OCTET STRING~~

-- For generic use

4.1.2.6 Notation for VisibleString type

VisibleString ::= VISIBLE STRING

-- IEC 646 – International Reference Version
without the “del”(coding 0x7F) character

4.1.2.7 Notation for BinaryDate type

BinaryDate ::= OctetString SIZE (7)

4.1.2.8 Notation for TimeOfDay type

TimeOfDay with date indication ::= OctetString SIZE (6)

TimeOfDay without date indication ::= OctetString SIZE (4)

4.1.2.9 Notation for TimeDifference type

TimeDifference with date indication ::= OctetString SIZE (6)

TimeDifference without date indication ::= OctetString SIZE (4)

4.1.2.10 Notation for Network Time type

Network Time ::= OctetString SIZE (8)

4.1.2.11 Notation for Network Time Difference type

Network Time Difference ::= OctetString SIZE (8)

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 be 0xff.

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 Floating32 and Floating64 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 IEC 60559.

For Floating32 the sign is encoded in bit 7 of the first octet. It is followed by the exponent starting from bit 6 of the first octet, and then the mantissa starting from bit 6 of the second octet.

For Floating64 the sign is encoded in bit 7 of the first octet. It is followed by the exponent starting from bit 6 of the first octet, and then the mantissa starting from bit 3 of the second octet.

4.2.1.6 Encoding of a Visible String value

The encoding of a variable length VisibleString 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 Encoding of an OctetString value

The encoding of a variable length OctetString 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 second octet, followed by each octet in turn up to and including the last octet as rightmost of the ContentsOctets.

4.2.1.8 Encoding of a BinaryDate value

The encoding of a BinaryDate value shall be primitive.

The Length field shall indicate as a binary number the number of octets in the BinaryDate value.

The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 8:

bits octets	7	6	5	4	3	2	1	0	
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	RSV	RSV	2^5	2^4	2^3	2^2	2^1	2^0	0...59 min
4	SU	RSV	RSV	2^4	2^3	2^2	2^1	2^0	0...23 hours
5	day of week			day of month					1...7 day of week
	2^2	2^1	2^0	2^4	2^3	2^2	2^1	2^0	1...31 day of month
6	RSV	RSV	2^5	2^4	2^3	2^2	2^1	2^0	1...12 months
7	RSV	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
msb									lsb

NOTE To avoid the Y2K problem the values of the year are from 0 to 50 for the years 2000 to 2050 and from 51 to 99 for the years 1951 to 1999. This is according to the British Standards Institution (BSi) DISC PD2000-1:1998 "A Definition of Year 2000 Conformance Requirements" Rule 3 (b).

Figure 8 – Coding of the data type BinaryDate

4.2.1.9 Encoding of a Time Of Day with and without date indication value

The encoding of a Time Of Day with and without date indication value shall be primitive.

The Length field shall indicate as a binary number the number of octets in the Time Of Day with and without date indication value.

The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 9:

bits octets	7	6	5	4	3	2	1	0	
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	number of ms 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 only with date indication
6	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
msb									lsb

Figure 9 – Encoding of Time Of Day value

4.2.1.10 Encoding of a Time Difference with and without date indication value

The encoding of a Time Difference with and without date indication value shall be primitive.

The Length field shall indicate as a binary number the number of octets in the Time Difference with and without date indication value.

The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 10:

bits octets	7	6	5	4	3	2	1	0	
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	ms
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	only with date indication
msb									lsb

Figure 10 – Encoding of Time Difference value

4.2.1.11 Encoding of a Network Time value

The encoding of a Network Time value shall be primitive.

The Length field shall indicate as a binary number the number of octets in the Time Difference value.

The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 11.

bits octets	7	6	5	4	3	2	1	0	
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Seconds since 1900-01-01 0.00,00 or since 2036-02-07 6.28,16 when time value less than 0x9dff4400.00000000
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}	Fraction Part of seconds one unit is $1/(2^{32})$ s
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	
msb									lsb

Figure 11 – Encoding of Network Time value

4.2.1.12 Encoding of a Network Time Difference value

The encoding of a Network Time Difference value shall be primitive.

The Length field shall indicate as a binary number the number of octets in the Time Difference value.

The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 12.

bits octets	7	6	5	4	3	2	1	0	
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Seconds as Integer32
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}	Fraction Part of seconds one unit is $1/(2^{32})$ s
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	
	msb				lsb				

Figure 12 – Encoding of Network Time Difference value

4.2.1.13 Encoding of a NULL value

The encoding of a NULL value shall be primitive.

The ContentsOctet shall be omitted.

4.2.1.14 Encoding of a NIL value

The encoding of a NIL value shall be primitive.

Each ContentsOctet shall be set to zero.

4.2.1.15 Encoding of an UUID

The encoding of this data type shall be according to Publication C706 of The Open Group.

NOTE The content of Publication C706 is DCE RPC V1.1.

4.2.2 Coding section related to common basic fields

4.2.2.1 Overview

The common fields of 4.2.2 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]. The value of the field SourceAddress shall be according to IEEE 802 MAC address, to IEEE 802.1D-2004, Clause 7, and to Table 7.

Table 7 – 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
MRP-PDU, LLDP-PDU, PTCP-PDU	The port MAC address is used

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

NOTE 2 The port MAC address is name Bridge Port individual MAC address in IEEE 802.1D-2004, 7.12.

EXAMPLE A device with n ports needs one interface MAC address and n port MAC addresses.

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 IEEE 802 MAC address.

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

4.2.2.3.2 DCP-PDUs

For DCP-PDUs, this field shall be coded according to Table 8. For all other DCP-PDUs multicast or broadcast addresses shall not be used.

Table 8 – DCP_MulticastMACAdd

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-00	With FrameID=0xFEFE used for DCP-Identify-ReqPDU
01-0E-CF	00-00-01	With FrameID=0xFEFC used for DCP-Hello-ReqPDU
01-0E-CF	00-00-02 – 00-00-FF	Reserved for other applications

4.2.2.3.3 PTCP-PDUs

For PTCP-PDUs, the value shall be set according to Table 9, Table 10, Table 11, Table 12, Table 13, Table 14, Table 15, and Table 16.

Table 9 – PTCP_MulticastMACAdd range 1

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-01-00 – 00-03-FF	Reserved for other applications

Table 10 – 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 clock synchronization
01-0E-CF	00-04-01	In conjunction with PTCP-AnnouncePDU and FrameID (=0xFF01) used for time synchronization
01-0E-CF	00-04-xx	In conjunction with PTCP-AnnouncePDU and FrameID (=0xFFxx) used for synchronization
01-0E-CF	00-04-1F	In conjunction with PTCP-AnnouncePDU and FrameID (=0xFF1F) used for synchronization

Table 11 – 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 clock synchronization
01-0E-CF	00-04-21	In conjunction with PTCP-RTSyncPDU with follow up and FrameID(=0x0021) used for time synchronization
01-0E-CF	00-04-xx	In conjunction with PTCP-RTSyncPDU with follow up, and FrameID(=0x00xx) used for synchronization
01-0E-CF	00-04-3F	In conjunction with PTCP-RTSyncPDU with follow up and FrameID(=0x003F) used for synchronization

Table 12 – 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 clock synchronization
01-0E-CF	00-04-41	In conjunction with PTCP-FollowUpPDU and FrameID(=0xFF21) used for time synchronization
01-0E-CF	00-04-xx	In conjunction PTCP-FollowUpPDU and FrameID(=0xFFxx) used for synchronization
01-0E-CF	00-04-5F	In conjunction with PTCP-FollowUpPDU and FrameID(=0xFF3F) used for synchronization

Table 13 – 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 14 – 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 clock synchronization
01-0E-CF	00-04-81	In conjunction with PTCP-RTSyncPDU and FrameID(=0x0081) used for time synchronization
01-0E-CF	00-04-xx	In conjunction with PTCP-RTSyncPDU and FrameID(=0x00xx) used for synchronization
01-0E-CF	00-04-9F	In conjunction with PTCP-RTSyncPDU and FrameID(=0x009F) used for synchronization

Table 15 – PTCP_MulticastMACAdd range 7

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-A0	00-04-40 – FF-FF-FF	Reserved for other applications

Table 16 – PTCP_MulticastMACAdd range 8

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

4.2.2.3.4 MRP-PDUs

The IEEE Organizationally Unique Identifier for MRP is 00-15-4E. It shall be set according to the Table 17.

Table 17 – MRP Organizationally Unique Identifier (OUI)

Value OUI (hexadecimal)	Meaning
00-15-4E	Global administered individual unicast
01-15-4E	Global administered group (multicast) address
02-15-4E	Local administered individual unicast
03-15-4E	Local administered group (multicast) address

For MRP-PDUs, the value shall be set according to the Table 18.

Table 18 – MRPMulticastMACAdd

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-15-4E	00-00-00	Reserved
01-15-4E	00-00-01	MC_Test, used for media redundancy test frames
01-15-4E	00-00-02	MC_CONTROL, used for media redundancy link change and topology change frames
01-15-4E	00-00-03 – FF-FF-FF	Reserved

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 19.

Table 19 – LT (Length/Type)

Value (hexadecimal)	Meaning
0x0800	IP (UDP, RPC, SNMP, ICMP)
0x0806	ARP
0x8100	Tag Control Information
0x8892	RTC, RTA, DCP, PTCP
0x88E3	MRP
0x88CC	LLDP

4.2.2.5 Coding of the field TagControlInformation

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 11: TagControlInformation.VLAN_Id

This field shall be coded according to IEEE 802.1Q.

NOTE It is recommended to use the VLAN_Id 0 that means no use of VLANs.

Bit 12: TagControlInformation.CanonicalFormatIdentifier

This field shall be coded according to IEEE 802.1Q.

NOTE CFI is constant 0.

Bit 13 – 15: TagControlInformation.Priority

This field shall be coded according to IEEE 802.1Q.

This field shall be coded with the values according to Table 20.

Table 20 – TagControlInformation.Priority

Value (hexadecimal)	Meaning
0x00	IP, DCP
0x01	Reserved
0x02	Reserved
0x03	Reserved
0x04	Reserved
0x05	Low prior RTA_CLASS_1or RTA_CLASS_UDP
0x06	RT_CLASS_UDP, RT_CLASS_1, RT_CLASS_2 high prior RTA_CLASS_1 or RTA_CLASS_UDP
0x07	Reserved

NOTE If the used IP implementation does not provide Tag Control Information then the RTA_CLASS_UDP and the RT_CLASS_UDP is send 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 21, Table 22, Table 23, Table 24, Table 25, Table 26, Table 27, Table 28, Table 29, Table 30, Table 31, and Table 32. This field identifies the structure and the type of the APDU.

Table 21 – 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 Send Clock and Phase Synchronization
0x0021	PTCP-RTSyncPDU (with follow up)	Precision transparent clock protocol synchronization with follow up Time Synchronization
0x0022 – 0x007F	Reserved	Reserved

Table 22 – FrameID range 2

Value (hexadecimal)	Meaning	Use
0x0080	PTCP-RTSyncPDU	Precision transparent clock protocol synchronization without follow up Send Clock and Phase Synchronization
0x0081	PTCP-RTSyncPDU	Precision transparent clock protocol synchronization without follow up Time Synchronization
0x0082 – 0x009F	Reserved	Reserved
0x00A0 – 0x00FF	Reserved	Reserved

Table 23 – FrameID range 3

Value (hexadecimal)	Meaning	Use
0x0100 – 0x04FF	Dedicated to RT_CLASS_3 (RED) unicast and multicast	RED, non redundant, normal
0x0500 – 0x05FF	Dedicated to RT_CLASS_3 (RED) unicast and multicast	Reserved
0x0600 – 0x07FF	Dedicated to RT_CLASS_3 (RED) unicast and multicast	Reserved
0x0800 – 0x0FFF	Dedicated to RT_CLASS_3 (RED) unicast and multicast	RED, redundant, normal ^a
0x1000 – 0x47FF	Reserved	Reserved

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

Table 24 – FrameID range 4

Value (hexadecimal)	Meaning	Use
0x4800 – 0x67FF	—	Reserved

Table 25 – FrameID range 5

Value (hexadecimal)	Meaning	Use
0x6800 – 0x7FFF	—	Reserved

Table 26 – FrameID range 6

Value (hexadecimal)	Meaning	Use
0x8000 – 0xBBFF	Dedicated to RT_CLASS_1 (GREEN) and RT_CLASS_2 (ORANGE) unicast	GREEN / ORANGE, non redundant, normal
0xBC00 – 0xBFFF	Dedicated to RT_CLASS_1 (GREEN) and RT_CLASS_2 (ORANGE) multicast	GREEN / ORANGE, non redundant, normal

Table 27 – FrameID range 7

Value (hexadecimal)	Meaning	Use
0xC000 – 0xF7FF	Dedicated to RT_CLASS_UDP unicast	RT_CLASS_UDP (recommended) and RT_CLASS_1(legacy); disjunct Frame IDs shall be used if concurrent usage in different ARs
0xF800 – 0xFBFF	Dedicated to RT_CLASS_UDP multicast	RT_CLASS_UDP (recommended) and RT_CLASS_1(legacy); disjunct Frame IDs shall be used if concurrent usage in different ARs

Table 28 – FrameID range 8

Value (hexadecimal)	Meaning	Use
0xFC00	Reserved	—
0xFC01	Alarm High	RTA_CLASS_1 and RTA_CLASS_UDP
0xFC02 – 0xFDFD	Reserved	—
0xFE00	Reserved	—
0xFE01	Alarm Low	RTA_CLASS_1 and RTA_CLASS_UDP
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 29 – FrameID range 9

Value (hexadecimal)	Meaning	Use
0xFF00	PTCP-AnnouncePDU (clock)	Precision transparent clock announce protocol Isochronous application, send clock, and phase synchronization
0xFF01	PTCP-AnnouncePDU (time)	Precision transparent clock announce protocol Time synchronization
0xFF02 – 0xFF1F	Reserved	—
0xFF20	PTCP-FollowUpPDU (clock)	Send Clock and phase synchronization (PTCP- FollowUpPDU)
0xFF21	PTCP-FollowUpPDU (time)	Time synchronization (PTCP-FollowUpPDU)
0xFF22 – 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 30 – FrameID range 10

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

Table 31 – FrameID range 11

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

Table 32 – FrameID range 12

Value (hexadecimal)	Meaning	Use
0xFF80 – 0xFF8F	—	Reserved

Table 33 – FrameID range 13

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

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 34 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 34 – DCP APDU syntax

APDU name	APDU structure
DCP-PDU	DCP-Get-ReqPDU ^ DCP-Set-ReqPDU ^ DCP-Get-ResPDU ^ DCP-Set-ResPDU ^ DCP-Identify-ReqPDU ^ DCP-Identify-ResPDU ^ DCP-Hello-ReqPDU
DCP-Identify-ReqPDU	DCP-IdentifyFilter-ReqPDU ^ DCP-IdentifyAll-ReqPDU
DCP-IdentifyFilter-ReqPDU	DCP-MC-Header, [NameOfStationBlock] ^ [AliasNameBlock], IdentifyReqBlock* ^a
DCP-IdentifyAll-ReqPDU	DCP-MC-Header, AllSelectorBlock
DCP-Hello-ReqPDU	DCP-UC-Header, NameOfStationBlockRes, { IPPParameterBlockRes, DeviceIDBlockRes, DeviceOptionsBlockRes, DeviceRoleBlockRes, DeviceInitiativeBlockRes }
DCP-Identify-ResPDU	DCP-UC-Header, { IdentifyResBlock * ^b , NameOfStationBlockRes ^c , IPPParameterBlockRes ^c , DeviceIDBlockRes ^c , DeviceOptionsBlockRes ^c , DeviceRoleBlockRes ^c , [DeviceInitiativeBlockRes] ^d }
DCP-Get-ReqPDU	DCP-UC-Header, GetReqBlock*
DCP-Get-ResPDU	DCP-UC-Header, (GetResBlock ^ GetNegResBlock)*
DCP-Set-ReqPDU	DCP-UC-Header, [StartTransactionBlock, BlockQualifier], [FactoryResetBlock, BlockQualifier] ^ SetReqBlock*, [StopTransactionBlock, BlockQualifier]
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.

^c The field shall only be present if it is not already part of the IdentifyResBlock.

^d The field shall only be present if the usage of DCP-Hello-ReqPDU is activated.

Table 35 defines structures for substitutions of elements of the APDU structures shown in Table 34.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Withdrawn

Table 35 – 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 ^ MACAddressBlock ^ IPPParameterBlock ^ DHCPParameterBlock ^ ManufacturerSpecificParameterBlock
IdentifyResBlock	NameOfStationBlockRes ^ DeviceRoleBlockRes ^ DeviceVendorBlockRes ^ DeviceIDBlockRes ^ DeviceOptionsBlockRes ^ MACAddressBlockRes ^ IPPParameterBlockRes ^ DHCPParameterBlockRes ^ ManufacturerSpecificParameterBlockRes ^ AliasNameBlockRes
GetReqBlock	NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ MACAddressType ^ IPPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType ^ AllSelectorType
GetResBlock	NameOfStationBlockRes ^ DeviceRoleBlockRes ^ DeviceVendorBlockRes ^ DeviceIDBlockRes ^ DeviceOptionsBlockRes ^ MACAddressBlockRes ^ IPPParameterBlockRes ^ DHCPParameterBlockRes ^ ManufacturerSpecificParameterBlockRes
GetNegResBlock	ControlOption, SuboptionResponse, DCPBlocklength, NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ MACAddressType ^ IPPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType, BlockError, [AddDataValue]
SetReqBlock	SignalType ^ NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ IPPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType, DCPBlocklength, BlockQualifier, SignalValue ^ NameOfStationValue ^ DeviceRoleValue ^ DeviceVendorValue ^ DeviceIDValue ^ DeviceOptionsValue ^ IPPParameterValue ^ DHCPParameterValue ^ ManufacturerSpecificParameterValue
SetResBlock	ControlOption, SuboptionResponse, DCPBlocklength, NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ IPPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType ^ SignalType ^ FactoryResetType ^ StartTransactionType ^ StopTransactionType, BlockError(=0), [AddDataValue]
SetNegResBlock	ControlOption, SuboptionResponse, DCPBlocklength, NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ IPPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType ^ SignalType ^ FactoryResetType ^ StartTransactionType ^ StopTransactionType, BlockError(<>0), [AddDataValue]
DeviceInitiativeType	DeviceInitiativeOption, DeviceInitiativeSuboption
DeviceInitiativeBlock	DeviceInitiativeType, DCPBlocklength, DeviceInitiativeValue
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

Substitution name	Structure
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
DeviceVendorType	DevicePropertiesOption, SuboptionDeviceVendor
DeviceVendorBlock	DeviceVendorType, DCPBlocklength, DeviceVendorValue
DeviceVendorBlockRes	DeviceVendorType, DCPBlocklength, BlockInfo, DeviceVendorValue
DeviceIDType	DevicePropertiesOption, SuboptionDeviceID
DeviceIDBlock	DeviceIDType, DCPBlocklength, DeviceIDValue
DeviceIDBlockRes	DeviceIDType, DCPBlocklength, BlockInfo, DeviceIDValue
DeviceIDValue	VendorIDHigh, VendorIDLow, DeviceIDHigh, DeviceIDLow
DeviceOptionsType	DevicePropertiesOption, SuboptionDeviceOptions
DeviceOptionsBlock	DeviceOptionsType, DCPBlocklength, DeviceOptionsValue
DeviceOptionsBlockRes	DeviceOptionsType, DCPBlocklength, BlockInfo, DeviceOptionsValue
DeviceOptionsValue	[NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ MACAddressType ^ IPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType]*
MACAddressType	IPOption, SuboptionMACAddress
MACAddressBlock	MACAddressType, DCPBlocklength, MACAddressValue
MACAddressBlockRes	MACAddressType, DCPBlocklength, BlockInfo, MACAddressValue
IPParameterType	IPOption, SuboptionIPParameter
IPParameterBlock	IPParameterType, DCPBlocklength, IPParameterValue
IPParameterBlockRes	IPParameterType, DCPBlocklength, BlockInfo, IPParameterValue
DHCPParameterType	DHCPOption, SuboptionDHCP
DHCPParameterBlock	DHCPParameterType, DCPBlocklength, DHCPParameterValue
DHCPParameterBlockRes	DHCPParameterType, DCPBlocklength, BlockInfo, DHCPParameterValue
ManufacturerSpecificParameterType	ManufacturerSpecificOption, SuboptionManufacturerSpecific
ManufacturerSpecificParameterBlock	ManufacturerSpecificParameterType, DCPBlocklength, ManufacturerSpecificParameterValue
ManufacturerSpecificParameterBlock Res	ManufacturerSpecificParameterType, DCPBlocklength, BlockInfo, ManufacturerSpecificParameterValue
AllSelectorType	AllSelectorOption, SuboptionAllSelector
AllSelectorBlock	AllSelectorType, DCPBlocklength
DeviceRoleValue	DeviceRoleDetails, Padding
IPParameterValue	IPAddress, Subnetmask, StandardGateway
DHCPParameterValue	SuboptionDHCP, DHCPParameterLength, DHCPParameterData
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 36. They shall be set in the appropriate PDU.

Table 36 – 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

This field shall be coded as data type Unsigned8 and shall contain the values as described in Table 37 and Table 38. They shall be set in the appropriate PDU.

Table 37 – ServiceType for request

Bit	Usage	Value (binary)	Meaning
Bit 0	Selection	0	Request
Bit 1 – 7	Reserved	0	Shall be set to zero

Table 38 – ServiceType for response

Bit	Usage	Value (binary)	Meaning
Bit 0	Selection	1	Response
Bit 1	Reserved	—	Shall be set to zero
Bit 2	Response	0	Success
		1	Request not supported
Bit 3 – 7	Reserved	—	Shall be set to zero

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 39. It shall contain a delay factor that shall be used by the server to calculate its individual response delay time. The response delay time is between 10 ms and 64 s.

Table 39 – 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 according to the following equations.

The last two octets of the IEEE 802 MAC address shall be used as random number K. Octet 6 of the MAC address represents the low-order octet and octet 5 the high-order octet.

$$\text{Spread} = K \bmod \text{ResponseDelayFactor} \quad (1)$$

Spread not equal zero:

$$\text{Minimal Response Delay} = 10 \text{ ms} \times \text{Spread} \quad (2)$$

Spread equal zero:

$$\text{Minimal Response Delay} = 0 \text{ ms} \quad (3)$$

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

ResponseDelayFactor equal one:

$$\text{Response delay timeout} = 400 \text{ ms} \quad (4)$$

ResponseDelayFactor greater than one:

$$\text{Intermediate value} = 1 \text{ s} + \text{ResponseDelayFactor} \times 10 \text{ ms} \quad (5)$$

$$\text{Response delay timeout} = \text{Round up to next second (Intermediate value)} \quad (6)$$

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 40 shows the list of available options and Table 41, Table 42, Table 43, Table 44, Table 45, Table 46, and Table 47 shows the list of available suboptions.

Table 40 – 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 41 – List of suboptions for option IPOption

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

Table 42 – 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	Read
Other	Reserved	—

Table 43 – List of suboptions for option DHCPOption

Value (hexadecimal)	Meaning	Accessible
See Table 48	SuboptionDHCP	Read / Write

Table 44 – List of suboptions for option ControlOption

Value (hexadecimal)	Meaning	Accessible
0x01	SuboptionStart	Write
0x02	SuboptionStop	Write
0x03	SuboptionSignal	Write
0x04	SuboptionResponse	—
0x05	SuboptionFactoryReset	Write
Other	Reserved	—

Table 45 – List of suboptions for option DeviceInitiativeOption

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

Table 46 – List of suboptions for option AllSelectorOption

Value (hexadecimal)	Meaning	Accessible
0xFF	SuboptionAllSelector	—
Other	Reserved	—

Table 47 – 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 40.

4.3.1.4.2.2 Coding of the field SuboptionMACAddress

This field shall be coded as data type Unsigned8 according to Table 41.

4.3.1.4.2.3 Coding of the field SuboptionIPParameter

This field shall be coded as data type Unsigned8 according to Table 41.

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 40.

4.3.1.4.3.2 Coding of the field SuboptionDeviceVendor

This field shall be coded as data type Unsigned8 according to Table 42.

4.3.1.4.3.3 Coding of the field SuboptionNameOfStation

This field shall be coded as data type Unsigned8 according to Table 42.

4.3.1.4.3.4 Coding of the field SuboptionDeviceID

This field shall be coded as data type Unsigned8 according to Table 42.

4.3.1.4.3.5 Coding of the field SuboptionDeviceRole

This field shall be coded as data type Unsigned8 according to Table 42.

4.3.1.4.3.6 Coding of the field SuboptionDeviceOptions

This field shall be coded as data type Unsigned8 according to Table 42.

4.3.1.4.3.7 Coding of the field SuboptionAliasName

This field shall be coded as data type Unsigned8 according to Table 42.

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 40.

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 Table 48.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 48 – SuboptionDHCP

Value (decimal)	Data Length	Description	References
0 – 11		Reserved ^a	—
12	1 +	Host Name ^d	RFC 2132
13 – 42		Reserved ^a	—
43	1 +	Vendor specific information	RFC 2132
44 – 53		Reserved ^a	—
54	4	Server identifier	RFC 2132
55	1 +	Parameter request list ^c	RFC 2132
56 – 59		Reserved ^a	—
60	1 +	Class-identifier	RFC 2132
61	2 +	DHCP client identifier	RFC 2132
62 – 80		Reserved ^a	—
81	1 +	FQDN, Fully Qualified Domain Name ^d	—
82 – 96		Reserved ^a	—
97	Variable	UUID/GUID-based Client (=addressed Station) Identifier	—
98 – 254		Reserved ^a	—
255	1	Control DHCP for address resolution 0: Don't use DHCP 1: Don't use DHCP and reset all DHCP options 2: Use DHCP ^b for address resolution	—

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

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

^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 40.

4.3.1.4.5.2 Coding of the field SuboptionStart

This field shall be coded as data type Unsigned8 according to Table 44 and with a DCPBlocklenth according to Table 49.

Table 49 – 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 44 and with a DCPBlocklength according to Table 50.

Table 50 – 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 44 and with a DCPBlocklength according to Table 51.

Table 51 – 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 44.

4.3.1.4.5.6 Coding of the field SuboptionFactoryReset

This field shall be coded as data type Unsigned8 according to Table 44 and with a DCPBlocklength according to Table 52.

Table 52 – 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.

Factory Reset shall permanent set

The NameOfStation to “”

The IP address, the subnet mask and the standard gateway to 0.0.0.0

All other parameters to the manufacturers default value.

NOTE Every device should support a possibility to execute “reset to factory” without an engineering device. This could be done with a switch or by other means.

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 40.

4.3.1.4.6.2 Coding of the field SuboptionDeviceInitiative

This field shall be coded as data type Unsigned8 according to Table 45 and with a DCPBlocklength according to Table 53.

Table 53 – 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 40.

4.3.1.4.7.2 Coding of the field SuboptionAllSelector

This field shall be coded as data type Unsigned8 according to Table 46.

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 40.

4.3.1.4.8.2 Coding of the field SuboptionManufacturerSpecific

This field shall be coded as data type Unsigned8 according to Table 47.

4.3.1.4.9 Coding of the field DCPBlocklength

This field shall be coded as data type Unsigned16 and shall contain the length of the specific data of the suboption without DCPBlocklength itself and without counting padding octets.

4.3.1.4.10 Coding of the field BlockQualifier

This field shall be coded as data type Unsigned16.

Allowed values with the option IP suboption IP parameter shall be according to Table 54.

Table 54 – BlockQualifier with option IP

Bit	Value (binary)	Meaning
Bit 0	0	Use IP Address, Subnetmask, and Default Gateway temporary and delete permanently store values Set permanent IP Address, Subnetmask, and Default Gateway to 0.0.0.0
	1	Save IP Address, Subnetmask, and Default Gateway permanent and use it after restart
Bit 1 – 15	—	Reserved

NOTE The IP suboption MAC Address is not topic of change via DCP and therefore no BlockQualifier is defined.

Allowed values with the option DeviceProperties, DHCP, and ManufacturerSpecific shall be according to Table 55.

Table 55 – BlockQualifier with option DeviceProperties, DHCP, and ManufacturerSpecific

Bit	Value (binary)	Meaning
Bit 0	0	Use the value of the option/suboption temporary
	1	Save the value of the option/suboption permanent and use it also after restart
Bit 1 – 15	—	Reserved

In all other cases the field BlockQualifier shall be set to zero.

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 56.

Table 56 – 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	If optional suboption is not supported
0x03	Suboption not set	If suboption could not be set for local reasons
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
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 57, Table 58, Table 59, and Table 60.

Table 57 – BlockInfo for SuboptionIPParameter

Bit	Meaning
Bit 0 – 1	See Table 58
Bit 2 – 6	Reserved
Bit 7	See Table 59
Bit 8 – 15	Reserved

Table 58 – Bit 1 and Bit 0 of BlockInfo for SuboptionIPParameter

Bit 1	Bit 0	Meaning for SuboptionIPParameter
0	0	No IP parameter
0	1	IP parameter set
1	0	IP parameter set via DHCP
1	1	Reserved

Table 59 – Bit 7 of BlockInfo for SuboptionIPParameter

Bit 7	Meaning for SuboptionIPParameter
0	IP address active No IP address conflict detected ^a
1	IP address not active IP address conflict detected ^a

^a A 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 60 – 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 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0: DeviceInitiativeValue.Hello

This field shall be set according to the Table 61.

Table 61 – DeviceInitiativeValue

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

Bit 1 – 15: DeviceInitiativeValue.reserved

This field shall be set according to 3.7.3.2.

4.3.1.4.14 Coding of the field SignalValue

This field shall be coded as data type Unsigned16 according to Table 62.

Table 62 – SignalValue

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

4.3.1.4.15 Coding of the field NameOfStationValue

4.3.1.4.15.1 Encoding

This field shall be coded as data type OctetString with 1 to 240 octets. 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 [-]

The first label does not start with "port-xyz" or "port-xyz-abcde" with a,b,c,d,e, x, y, z = 0...9

Station-names do not have the form n.n.n.n, n = 0...999

Labels do only start with 'xn-' if RFC 3490 is applied

Furthermore, the definition of RFC 3490 shall be applied.

EXAMPLE 1 "device-1.machine-1.plant-1.vendor"

EXAMPLE 2 "device-1.bögeholz" is coded as "device-1.xn-bgeholz-90a"

NOTE The field NameOfStationValue is not terminated by zero.

Each NameOfStation shall be unique in case a device is equipped with more than one interface module.

4.3.1.4.15.2 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 4.3.1.4.15.1.

4.3.1.4.15.3 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 4.3.1.4.15.1.

4.3.1.4.16 Coding of the field AliasNameValue

4.3.1.4.16.1 Encoding

The field shall be coded as OctetString. The content shall be the concatenation of the content of the fields LLDP_PortID and LLDP_ChassisID.

AliasNameValue = LLDP_PortID + "." + LLDP_ChassisID (7)

4.3.1.4.16.2 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).

Name of the station with length == 0:

Not allowed in a request.

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

4.3.1.4.17 Coding of the field DeviceRoleDetails

This field shall be coded as data type Unsigned8 with the values according to Table 63.

Table 63 – DeviceRoleDetails

Bit	Meaning	Usage
Bit 0	PNIO Device	Device has the specific role
Bit 1	PNIO Controller	Device has the specific role
Bit 2	PNIO Multidevice	Device has the specific role
Bit 3	PNIO Supervisor	Device has the specific role
Bit 4 – 7	Reserved	—

4.3.1.4.18 Coding of the field MACAddressValue

This field shall be coded as data type OctetString[6]. The value of the field shall be according to IEEE 802 MAC address.

4.3.1.4.19 Coding of the field IPAddress

This field shall be coded as data type OctetString[4]. The encoding of the fields shall be according to RFC 791, RFC 3330 and to Table 64.

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

Table 64 – IPAddress

Value (CIDR ^a)	Meaning
0.0.0.0 / 8	No IP address assigned
127.0.0.0 / 8	Reserved Loopback
224.0.0.0 / 4	Reserved Multicast
Other	IP address assigned
Reserved	If a subnet is used, the top most IPAddress of this subnet is used as a subnet broadcast address. The assignment of this IPAddress shall be rejected.
^a Notation according to classless inter domain routing defined in IETF RFC 1518 and IETF RFC 1519	

4.3.1.4.20 Coding of the field Subnetmask

This field shall be coded as data type OctetString[4]. The encoding of the fields shall be according to RFC 791, RFC 3330 and to Table 65.

Table 65 – Subnetmask

Value	Number of hosts	Meaning
255.255.255.255	1	Subnet mask assigned
255.255.255.254	2	
255.255.255.252	4	
255.255.255.248	8	
255.255.255.240	16	
255.255.255.224	32	
255.255.255.192	64	
255.255.255.128	128	
255.255.255.000	256	
255.255.254.000	512	
255.255.252.000	1024	
255.255.248.000	2048	
255.255.240.000	4096	
255.255.224.000	8192	
255.255.192.000	16384	
255.255.128.000	32768	
255.255.000.000	65536	
255.254.000.000	131072	
255.252.000.000	262144	
255.248.000.000	524288	
255.240.000.000	1048576	
255.224.000.000	2097152	Subnet mask valid, support optional according to the rules of IANA
255.192.000.000	4194304	
255.128.000.000	8388608	
255.000.000.000	16777216	
254.000.000.000	33554432	Subnet mask valid, but rejected according to the rules of IANA
252.000.000.000	67108864	
248.000.000.000	134217728	
240.000.000.000	268435456	
224.000.000.000	536870912	Subnet mask valid, but rejected according to the rules of IANA
192.000.000.000	1073741824	
128.000.000.000	2147483648	
000.000.000.000	4294967296	Subnet mask valid, but rejected according to the rules of IANA if a StandardGateway is assigned. Subnet mask valid and assigned if no StandardGateway is assigned.
Other	Reserved	Invalid subnet mask

4.3.1.4.21 Coding of the field StandardGateway

This field shall be coded as data type OctetString[4]. The encoding of the field shall be according to RFC 791, RFC 3330, Table 64 and Table 66.

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

Table 66 – StandardGateway

Value (CIDR ^a)	Meaning
0.0.0.0 / 32	No standard gateway assigned The application shall set up the local IP implementation to avoid IP routing
Own IPAddress / 32	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 1518 and IETF RFC 1519 ^b Valid according to Table 64 and Table 65 ^c Invalid according to Table 64 and Table 65	

4.3.1.4.22 Coding of the fields DHCPParameterValue using DHCP Option 61

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

4.3.1.4.22.1 Coding of the field DHCPParameterLength

This field shall be coded as data type Unsigned16 and shall contain the length of the specific subsequent data.

4.3.1.4.22.2 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 following values:

SuboptionDHCP = 61

DHCPParameterLength = 1

DHCPParameterData = 1

4.3.1.4.22.3 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 following values (in addition to RFC 2132):

SuboptionDHCP = 61

DHCPParameterLength = 1

DHCPParameterData = 0

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

4.3.1.4.22.4 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 following values:

SuboptionDHCP	= 61
DHCPPParameterLength	=> 1
DHCPPParameterData[first octet]	= 0
DHCPPParameterData[second octet ...n]	= "NameOfStation"

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 DHCPPParameterData

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.

4.3.2 DCP protocol state machines

4.3.2.1 Application relationship monitoring

4.3.2.1.1 Timer

UC Client Timeout

This Timer, restarted on the requestor when sending an unicast request, will terminate waiting for an response. The value shall be taken from the appropriate DCP ASE attribute.

MC Client Timeout

This Timer, restarted when sending a multicast request on the requestor, will terminate waiting for a response. The value shall be taken from the appropriate DCP ASE attribute.

4.3.2.1.2 Intervals of the receiver

Response Delay Time

This monitoring time specifies the time period waiting for DCP-IdentifyResPDUs.

Client Hold Time

This monitoring time specifies the smallest allowed period of time between a response and the release of the client. This is used to allow multiple services executed from a single source without disruption of other nodes. The value shall be taken from the appropriate DCP ASE attribute.

4.3.2.2 Application Relationship Protocol Machines (ARPMs)

4.3.2.2.1 DCPUCS

4.3.2.2.1.1 Primitive definitions

4.3.2.2.1.1.1 Primitives exchanged between DCPUCS and DCP user

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

4.3.2.2.1.1.2 Primitives exchanged between DCPUCS and LMPM

The service primitives including their associated parameters issued by DCPUCS received by LMPM and vice versa are described in the LMPM service definition.

4.3.2.2.1.2 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.

Local variables of the DCPUCS:

SXID

This local variable contains the Transaction ID XID used to identify services uniquely. The initial value shall be derived from a random number generator.

NOTE The current time value in nanoseconds may be used as a random number.

Pending

This local variable holds the information whether a Set or Get service is pending.

Retry

This local variable contains the retry counter. The maximum value shall be taken from the appropriate DCP ASE attribute.

4.3.2.2.1.3 DCPUCS state table

Table 67 contains the complete description of the DCPUCS state table.

Table 67 – DCPUCS state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	DCP_Get.req(CREP, DA, ListOfOptions) => Pending:= GET DCP-UC-Header.Xid:= SXID A_SDU := DCP-GetReqPDU Store A_SDU Retry:= Max Retry Limit StartTimer(UC Client Timeout) LMPM_A_Data.req (CREP, DA, SA, Prio=0, A_SDU)	WACK
2	OPEN	DCP_Set.req(CREP, DA, ListOfData, ListOfControlCommands) => Pending:= SET DCP-UC-Header.Xid:= SXID A_SDU := DCP-SetReqPDU Store A_SDU Retry:= Max Retry Limit StartTimer(UC Client Timeout) LMPM_A_Data.req (CREP, DA, SA, Prio=0, A_SDU)	WACK
3	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) => ignore	OPEN
4	OPEN	LMPM_A_Data.cnf (CREP, LMPM_status) => ignore	OPEN
5	OPEN	Timeout => ignore	OPEN
6	WACK	DCP_Get.req(CREP, DA, ListOfOptions) => ERRCLS := CTXT ERRCODE := INVALID_STATE DCP_Get.cnf(-) (ERRCLS, ERRCODE)	WACK
7	WACK	DCP_Set.req(CREP, DA, ListOfData, ListOfControlCommands) => ERRCLS := CTXT ERRCODE := INVALID_STATE DCP_Set.cnf(-) (ERRCLS, ERRCODE)	WACK

#	Current State	Event /Condition =>Action	Next State
8	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /DCP-Get-ResPDU && Pending == GET && DCP-Header.xid == SXID => SXID:= (SXID+1) & 0xffffffff DCP_Get.cnf(+) (ListOfData)	OPEN
9	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /DCP-Get-ResPDU && (Pending != GET DCP-Header.xid != SXID) => ignore	WACK
10	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /DCP-Set-ResPDU && Pending == SET && DCP-Header.xid == SXID => SXID:= (SXID+1) & 0xffffffff DCP_Set.cnf(+) (CREP, ListOfData, ListOfControlCommands)	OPEN
11	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /DCP-Set-ResPDU && (Pending != SET DCP-Header.xid != SXID) => ignore	WACK
12	WACK	LMPM_A_Data.cnf (CREP, LMPM_status) /LMPM_status == OK => ignore	WACK
13	WACK	LMPM_A_Data.cnf (CREP, LMPM_status) /LMPM_status != OK && Pending == GET => ERRCLS := PROTOCOL ERRCODE := LMPM StopTimer() DCP_Get.cnf(-) (ERRCLS, ERRCODE)	CLOSED
14	WACK	LMPM_A_Data.cnf (CREP, LMPM_status) /LMPM_status != OK && Pending == SET => ERRCLS := PROTOCOL ERRCODE := LMPM StopTimer() DCP_Set.cnf(-) (ERRCLS, ERRCODE)	CLOSED

#	Current State	Event /Condition =>Action	Next State
15	WACK	Timeout /Retry != 0 => A_SDU:=from Stored A_SDU Retry:= Retry-1 LMPM_A_Data.req (DA, SA, Prio, A_SDU)	WACK
16	WACK	Timeout /Retry == 0 && Pending == GET => ERRCLS := PROTOCOL ERRCODE := TIMEOUT DCP_Get.cnf(-) (ERRCLS, ERRCODE)	OPEN
17	WACK	Timeout /Retry == 0 && Pending == SET => ERRCLS := PROTOCOL ERRCODE := TIMEOUT DCP_Set.cnf(-) (ERRCLS, ERRCODE)	OPEN
18	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) !(DCP-Set-ResPDU DCP-Get-ResPDU) => ignore	WACK

4.3.2.2.2 DCPUCR

4.3.2.2.2.1 Primitive definitions

4.3.2.2.2.1.1 Primitives exchanged between DCPUCR and DCP user

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

4.3.2.2.2.1.2 Primitives exchanged between DCPUCR and LMPM

The service primitives including their associated parameters issued by DCPUCS received by LMPM and vice versa are described in the LMPM ASE in the service definition.

4.3.2.2.2.2 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.

Local variables of the DCPUCR:

SAM

This local variable contains the last valid SA conveyed with a LMPM_A_Data indication with valid data.

SXID

This local variable contains the Transaction ID XID used to identify services uniquely. The value shall be taken from the request PDU.

4.3.2.2.2.3 DCPUCR state table

Table 68 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.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 68 – DCPUCR state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	DCP_Get.rsp(CREP, ListOfData) => ignore	OPEN
2	OPEN	DCP_Set.rsp(CREP, ListOfResponse) => ignore	OPEN
3	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /DCP-Get-ReqPDU && (SAM == NIL SAM == SA) => SAM:= SA SXID := DCP_Header.xid StartTimer(Client Hold Time) DCP_Get.ind (SA,ListOfSelectors)	WACK
4	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /DCP-Set-ReqPDU && (SAM == NIL SAM == SA) => SAM:= SA SXID := DCP_Header.xid StartTimer(Client Hold Time) DCP_Set.ind (SA,ListOfData)	WACK
5	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /DCP-Get-ReqPDU && (SAM != NIL && SAM != SA) => ignore	OPEN
6	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /DCP-Set-ReqPDU && (SAM != NIL && SAM != SA) => ignore	OPEN
7	OPEN	LMPM_A_Data.cnf (CREP, LMPM_status) => ignore	OPEN
8	OPEN	Timeout => SAM:= NIL	OPEN

#	Current State	Event /Condition =>Action	Next State
9	WACK	DCP_Get.rsp(CREP, ListOfData) => DCP_Header.op:= GET DCP_Header.xid:= SXID DA := SAM A_SDU := DCP-Get-ResPDU StopTimer() LMPM_A_Data.req (CREP, DA, SA, Prio, A_SDU)	OPEN
10	WACK	DCP_Set.rsp(CREP, ListOfResponse) => DCP_Header.op:= SET DCP_Header.xid:= SXID DA := SAM A_SDU := DCP-Set-ResPDU StopTimer() LMPM_A_Data.req (CREP, DA, SA, Prio, A_SDU)	OPEN
11	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) => ignore	WACK

4.3.2.2.3 DCPMCS

4.3.2.2.3.1 Primitive definitions

4.3.2.2.3.1.1 Primitives exchanged between DCPMCS and DCP user

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

4.3.2.2.3.1.2 Primitives exchanged between DCPMCS and LMPM

The service primitives including their associated parameters issued by DCPMCS received by LMPM and vice versa are described in the LMPM service definition.

4.3.2.2.3.2 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.

Local variables of the DCPMCS

SXID

This local variable contains the Transaction ID XID used to Identify services uniquely. The initial value shall be derived from a random number generator.

4.3.2.2.3.3 DCPMCS state table

Table 69 contains the complete description of the DCPMCS state machine. A LMPM_A_Data.ind primitive shall be accepted if type is DCP-IdentifyResPDU.

Table 69 – DCPMCS state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	DCP_Identify.req(ListOfFilter, ResponseDelayFactor, isASU) => DCP_Header.xid:= SXID DA=01:0E:CF:00:00:00 A_SDU := DCP-Identify-ReqPDU StartTimer(Response delay timeout) First := TRUE LMPM_A_SDU.req(CREP,DA, SA, VLAN-Prio,A_SDU)	WACK
2	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) => ignore	OPEN
3	OPEN	LMPM_A_Data.cnf (CREP, LMPM_status) => ignore	OPEN
4	OPEN	Timeout => ignore	OPEN
5	WACK	DCP_Identify.req(ListOfFilter, ResponseDelayFactor) => ERRCLS := CTXT ERRCODE := INVALID_STATE DCP_Identify.cnf(-) (CREP,ERRCLS,ERRCODE)	WACK
6	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /(DCP-Identify-ResPDU && DCP-Header.xid == SXID) && !(isASU && First) => ListOfDevices.SA += SA ListOfDevices.ListOfData += A_SDU First := FALSE	WACK
7	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /(DCP-Identify-ResPDU && DCP-Header.xid == SXID) && (isASU && First) => ListOfDevices.SA += SA ListOfDevices.ListOfData += A_SDU First := FALSE DCP_Identify.ind (SA, ListOfDevices)	WACK
8	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /(DCP-Identify-ResPDU && DCP-Header.xid == SXID) => ignore	WACK
9	WACK	LMPM_A_Data.cnf (CREP, LMPM_status) /LMPM_status == OK => ignore	WACK
10	WACK	LMPM_A_Data.cnf (CREP, LMPM_status) /!(LMPM_status == OK) => ERRCLS := PROTOCOL ERRCODE := LMPM DCP_Identify.cnf(-) (ERRCLS,ERRCODE)	OPEN
11	WACK	Timeout => SXID:= SXID + 1 mod 0xffffffff ListOfDevices=Stored ListOfDevices DCP_Identify.cnf(+) (ListOfDevices)	OPEN

4.3.2.2.4 DCPMCR

4.3.2.2.4.1 Primitive definitions

4.3.2.2.4.1.1 Primitives exchanged between DCPMCR and DCP user

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

4.3.2.2.4.1.2 Primitives exchanged between DCPMCR and LMPM

The service primitives including their associated parameters issued by DCPMCR received by LMPM and vice versa are described in the LMPM ASE in the service definition.

4.3.2.2.4.2 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.

Local variables of the DCPMCR

SXID

This local variable contains the Transaction ID XID used to identify services uniquely.

4.3.2.2.4.3 DCPMCR state table

Table 70 contains the complete description of the DCPMCR state machine. A LMPM_A_Data.ind primitive shall be accepted if Type is DCP-IdentifyReqPDU.

Table 70 – DCPMCR state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	DCP_Identify.rsp(ListOfData) => ignore	OPEN
2	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /DCP-Identify-ReqPDU && Successful comparison with ListOfFilters => ResponseDelay :=((SA[4]*256+SA[5]) mod ResponseDelayFactor) *10 ms StartTimer(ResponseDelay) DCP_Identify.ind(ListOfFilter)	WRSP
3	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) /!DCP-Identify-ReqPDU => ignore	OPEN
4	OPEN	LMPM_A_Data.cnf (CREP, LMPM_status) => ignore	OPEN
5	WRSP	Timeout /Stored A_SDU => A_SDU=Stored A_SDU LMPM_A_Data.req (CREP, DA, SA, Prio, A_SDU)	OPEN
6	WRSP	Timeout /!Stored A_SDU => ignore	OPEN
7	WRSP	DCP_Identify.rsp(ListOfData) => DCP_Header..op:= IDENT DCP_Header.xid:= SXID A_SDU := List of Data Store A_SDU	WRSP
8	WRSP	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) => ignore	WRSP

#	Current State	Event /Condition =>Action	Next State
9	WRSP	LMPM_A_Data.cnf (CREP, LMPM_status) => ignore	WRSP

4.3.2.2.5 DCPHMCS

4.3.2.2.5.1 Primitive definitions

4.3.2.2.5.1.1 Primitives exchanged between DCPHMCS and DCP user

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

4.3.2.2.5.1.2 Primitives exchanged between DCPHMCS and LMPM

The service primitives including their associated parameters issued by DCPHMCS received by LMPM and vice versa are described in the LMPM service definition.

4.3.2.2.5.2 State machine description

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

Local variables of the DCPHMCS

SXID

This local variable contains the Transaction ID XID used to Identify services uniquely. The initial value shall be derived from a random number generator.

4.3.2.2.5.3 DCPHMCS state table

Table 71 contains the complete description of the DCPHMCS state machine. A LMPM_A_Data.ind primitive shall be accepted if type is DCP-Hello-ReqPDU.

Table 71 – DCPHMCS state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	DCP_Hello.req (ListOfData) => DCP_Header.xid:= SXID DA=01:0E:CF:00:00:01 A_SDU := DCP-Hello-ReqPDU LMPM_A_SDU.req (CREP,DA, SA, VLAN-Prio, A_SDU)	WACK
2	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) => ignore	OPEN
3	OPEN	LMPM_A_Data.cnf (CREP, LMPM_status) /LMPM_status == OK => ignore	OPEN
4	OPEN	LMPM_A_Data.cnf (CREP, LMPM_status) /LMPM_status != OK => ignore	OPEN
5	WACK	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) => ignore	WACK
6	WACK	LMPM_A_Data.cnf (CREP, LMPM_status) /LMPM_status == OK => DCP_Hello.cnf (+)	OPEN
7	WACK	LMPM_A_Data.cnf (CREP, LMPM_status) /LMPM_status != OK => ERRCLS := PROTOCOL ERRCODE := LMPM DCP_Hello.cnf (-) (ERRCLS,ERRCODE)	OPEN

4.3.2.2.6 DCPHMCR**4.3.2.2.6.1 Primitive definitions****4.3.2.2.6.1.1 Primitives exchanged between DCPHMCR and DCP user**

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

4.3.2.2.6.1.2 Primitives exchanged between DCPHMCR and LMPM

The service primitives including their associated parameters issued by DCPHMCR received by LMPM and vice versa are described in the LMPM ASE in the service definition.

4.3.2.2.6.2 State machine description

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

4.3.2.2.6.3 DCPHMCR state table

Table 74 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 72 – DCPHMCR state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) / DCP-Hello-ReqPDU && Successful comparison with ListOfData => DCP_Hello.ind (ListOfData)	OPEN
2	OPEN	LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU) / !DCP-Hello-ReqPDU => ignore	OPEN
3	OPEN	LMPM_A_Data.cnf (CREP, LMPM_status) => ignore	OPEN

4.4 Precision 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 73 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 73 – 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 Shall be existent, if initially transmitted in the RED period and may be omitted, if initially transmitted in the ORANGE or GREEN period

Table 74 defines structures for substitutions of elements of the APDU structures.

Table 74 – PTCP substitutions

Substitution name	Structure
PTCP-Header-Sync	PTCP_reserved_1, PTCP_reserved_2, PTCP_Delay10ns ^e , PTCP_SequenceID, PTCP_Delay1ns_Byte ^d , Padding ^a , PTCP_Delay1ns
PTCP-Header-FUP	[HWPadding*] ^b , PTCP_SequenceID, [Padding*] ^c , PTCP_Delay1ns_FUP
PTCP-Header-Announce	[HWPadding*] ^b , PTCP_SequenceID, [Padding*] ^e
PTCP-Header-Delay	[HWPadding*] ^b , PTCP_SequenceID, [Padding*] ^c , PTCP_Delay1ns
PTCPSubdomain	PTCP_TLVHeader, PTCP_MasterSourceAddress, PTCP_SubdomainUUID
PTCPTime	PTCP_TLVHeader, PTCP_EPOCHNumber, PTCP_Seconds, PTCP_NanoSeconds
PTCPTimeExtension	PTCP_TLVHeader, PTCP_Flags, PTCP_CurrentUTCOffset, [Padding*] ^a
PTCPMaster	PTCP_TLVHeader, PTCP_MasterPriority1, PTCP_MasterPriority2, PTCP_ClockClass, PTCP_ClockAccuracy, PTCP_ClockVariance, [Padding*] ^a
PTCPDelayParameter	PTCP_TLVHeader, PTCP_PortMACAddress
PTCPPortParameter	PTCP_TLVHeader, [Padding*] ^a , PTCP_T2PortRxDelay, PTCP_T3PortTxDelay
PTCPPortTime	PTCP_TLVHeader, [Padding*] ^a , PTCP_T2TimeStamp
PTCPOption	PTCP_TLVHeader, PTCP_Option, PTCP_SubType, Data*, [Padding*] ^a
PTCPEnd	PTCP_TLVHeader(0)
^a 32bit 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 3.7.3.4. 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 75.

Table 75 – PTCP_TLVHeader.Type

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

4.4.1.3.5 Coding section related to PTCP-Header

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.

4.4.1.3.5.1 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 76.

Table 76 – 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

4.4.1.3.5.2 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 3.7.3.3 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 77.

Table 77 – 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 3.7.3.2.

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

4.4.1.3.5.3 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 78.

Table 78 – 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

4.4.1.3.5.4 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 79.

Table 79 – PTCP_Delay1ns_FUP

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Nanosecond part of the propagation delay time

4.4.1.3.5.5 Coding of the field PTCP_SequenceID

This field shall be coded as data type Unsigned16 with the values according to Table 80.

Table 80 – PTCP_SequenceID

Meaning	Usage
Sync frame	The PTCP master increments these 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 these 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.

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 81.

Table 81 – 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. The value contains the time (resolution of 1 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 82.

Table 82 – PTCP_NanoSeconds

Value (hexadecimal)	Meaning
0x00000000 – 0x3B9AC9FF	Delay 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 IEEE 802 MAC address.

4.4.1.4.2 Coding of the field PTCP_SubdomainUUID

This field shall be coded as data type UUID.

4.4.1.4.3 Coding of the field PTCP_Flags

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: PTCP_Flags.reserved_1

This field shall be set according to 3.7.3.2.

Bit 8 – 9: PTCP_Flags.LeapSecond

This field shall be set according Table 83.

Table 83 – PTCP_Flags.LeapSecond

Value (hexadecimal)	Meaning
0x00	No leap second
0x01	Last minute has 61 seconds
0x02	Last minute has 59 seconds
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 global time properties data set of the clock issuing this message according to Table 84 and Table 85.

Table 84 – Timescale correspondence between MJD, UTC, and PTCP_EPOCHNumber

MJD (Modified Julian Day)	UTC (Universal Time Coordinated)	UTC Offset (Leap seconds)	Value PTCP_Seconds	Value PTCP_EPOCH- Number
0:00:00 15 020	1 January 1900 – 00:00:00	0	—	0
0:00:00 40 587	1 January 1970 – 00:00:00		0	0
0:00:00 41 317	1 January 1972 – 00:00:00	10	63 072 000 + 10	0
16:57:44 51 357	28 June 1999 – 16:57:44	32	930 589 064 + 32	0
	1 January 2006 – 00:00:00	33	—	0

Table 85 – 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
0x0000000000001000	0	65	536 000 000	32
0x0000000010000000	0	4 294 967	296 000 000	32
0x0001000000000000	65	2 302 102 470	656 000 000	32

$$\text{PTCP\_Epoch} = \text{PTCP\_EpochNumber} \times 0x100000000 \quad (8)$$

$$\text{PTCP\_Time} = \text{PTCP\_Epoch} + \text{PTCP\_Second} + \text{PTCP\_Nanosecond} \quad (9)$$

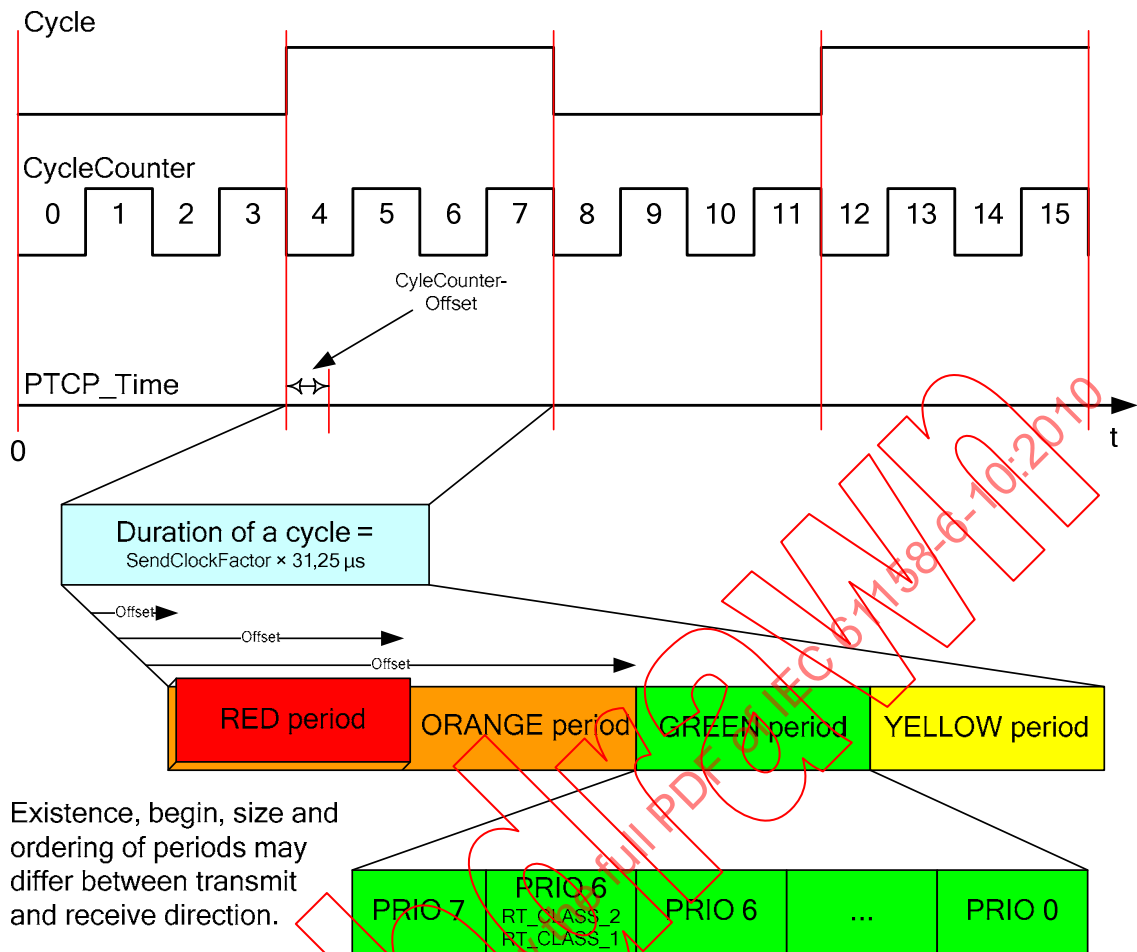


Figure 13 – Timescale correspondence between PTCP_Time and CycleCounter

For the synchronisation of the CycleCounter as shown in Figure 13, the calculation shall accept PTCP_Time values with an offset to the cycle begin.

$$\text{Cycle} = \text{PTCP\_Time} \text{ DIV } (\text{SendClockFactor} \times 31,25 \mu\text{s}) \quad (10)$$

$$\text{CycleCounter} = \text{Cycle} \times \text{SendClockFactor} \quad (11)$$

$$\text{CycleCounterOffset} = \text{PTCP\_Time} \text{ MOD } (\text{SendClockFactor} \times 31,25 \mu\text{s}) \quad (12)$$

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. The value of the current PTCP_CurrentUTCOffset field shall be the value of current_utc_offset of the global time properties data set of the clock issuing this message according to Table 84.

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 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0 – 2: PTCP_MasterPriority1.Priority

This field shall be set according to Table 86, Table 87, Table 88, and Table 89.

Table 86 – PTCP_MasterPriority1.Priority for SyncID == 0 and SyncProperties.Role == 2

Value (hexadecimal)	Usage	Meaning
0x00	Reserved	—
0x01	Primary synchronization master	Clock synchronization
0x02	Secondary synchronization master	Clock synchronization
0x03 – 0x07	Reserved	—

Table 87 – PTCP_MasterPriority1.Priority for SyncID == 0 and SyncProperties.Role == 1

Value (hexadecimal)	Usage	Meaning
0x00	Sync slave	Sync slave
0x01 – 0x07	Reserved	—

Table 88 – PTCP_MasterPriority1.Priority for SyncID == 1 and SyncProperties.Role == 2

Value (hexadecimal)	Usage	Meaning
0x00	Reserved	—
0x01	Primary synchronization master	Time synchronization
0x02	Secondary synchronization master	Time synchronization
0x03 – 0x07	Reserved	—

Table 89 – PTCP_MasterPriority1.Priority for SyncID == 1 and SyncProperties.Role == 1

Value (hexadecimal)	Usage	Meaning
0x00	Sync slave	Sync Slave
0x01 – 0x07	Reserved	—

Bit 3 – 5: PTCP_MasterPriority1.Level

This field shall be set according to Table 90.

Table 90 – PTCP_MasterPriority1.Level

Value (hexadecimal)	Usage	Meaning
0x00	Level 0 (highest)	See Annex N
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 and shall be set to one for the PTCP-SYNC-PDU. 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.

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 91.

Table 91 – 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 92 and Table 93. For clock synchronization the the arbitrary timescale (ARB) shall be used.

Table 92 – PTCP_ClockClass for SyncID == 0 (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 ARB. A clock class 0x0D clock shall not be a slave to another clock in the PTP 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 ARB. A clock class 0x0E clock shall not be a slave to another clock in the PTP domain.	—
0x0F – 0xBA	Reserved	—
0xBB	Degradation alternative B for a PTCP clock of clock class 0x07 that is not within holdover specification. A clock of clock class 0xBB may be a slave to another clock in the PTP 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 PTP domain.	—
0xC2 – 0xFE	Reserved	—
0xFF	Shall be the clock class of a slave-only clock.	—

Table 93 – PTCP_ClockClass for SyncID == 1 (time synchronization)

Value (hexadecimal)	Usage	Meaning
0x00 – 0x05	Reserved	—
0x06	Shall designate a clock as a primary reference that is synchronized to a recognized standard time source. The timescale distributed shall be PTCP. A clock class 0x06 clock shall not be a slave to another clock in the PTCP domain.	Time synchronization
0x07	Shall designate a PTCP clock that has previously been designated as clock class 0x06 but that has lost the ability to synchronize to a recognized standard time source and is in holdover mode and within holdover specifications. The timescale distributed shall be PTCP. A clock class 7 clock shall not be a slave to another clock in the PTCP domain.	Time synchronization
0x08 – 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 ARB. A clock class 0x0D clock shall not be a slave to another clock in the PTP domain.	Time synchronization
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 ARB. A clock class 0x0E clock shall not be a slave to another clock in the PTP domain.	Time synchronization
0x0F – 0x33	Reserved	—
0x34	Degradation alternative A for a PTCP clock of clock class 0x07 that is not within holdover specification. A clock of clock class 0x34 shall not be a slave to another clock in the PTP domain.	Time synchronization
0x35 – 0x39	Reserved	—
0x3A	Degradation alternative A for a PTCP clock of clock class 0x0E that is not within holdover specification. A clock of clock class 0x3A shall not be a slave to another clock in the PTP domain.	Time synchronization
0x3B – 0xBA	Reserved	—
0xBB	Degradation alternative B for a PTCP clock of clock class 0x07 that is not within holdover specification. A clock of clock class 0xBB may be a slave to another clock in the PTP domain.	Time synchronization
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 PTP domain.	Time synchronization
0xC2 – 0xFE	Reserved	—
0xFF	Shall be the clock class of a slave-only clock.	Time synchronization

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 94. 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 94 – 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	Default for clock and time synchronisation
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	—
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 95. 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 95 – PTCP_ClockVariance

Value (hexadecimal)	Usage	Meaning
0x0000 – 0xFFFF	Reserved	—

4.4.1.4.11 Coding of the field PTCP_T2PortRxDelay

This field shall be coded as data type Unsigned32. The value contains the port-receive-delay in 1 ns.

4.4.1.4.12 Coding of the field PTCP_T3PortTxDelay

This field shall be coded as data type Unsigned32. The value contains the port-transmit-delay in 1 ns.

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 IEEE 802 MAC address.

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 96.

Table 96 – 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 (FSPMs)

There is no FAL Service Protocol Machine defined for this Protocol.

4.4.4 Application Relationship Protocol Machines (ARPMs)

4.4.4.1 Generic Synchronization with PTCP

4.4.4.1.1 Timestamp

For each received and transmitted sync, delay request or delay response message a time stamping unit has to generate a time stamp. The PTCP message timestamp point 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 14.

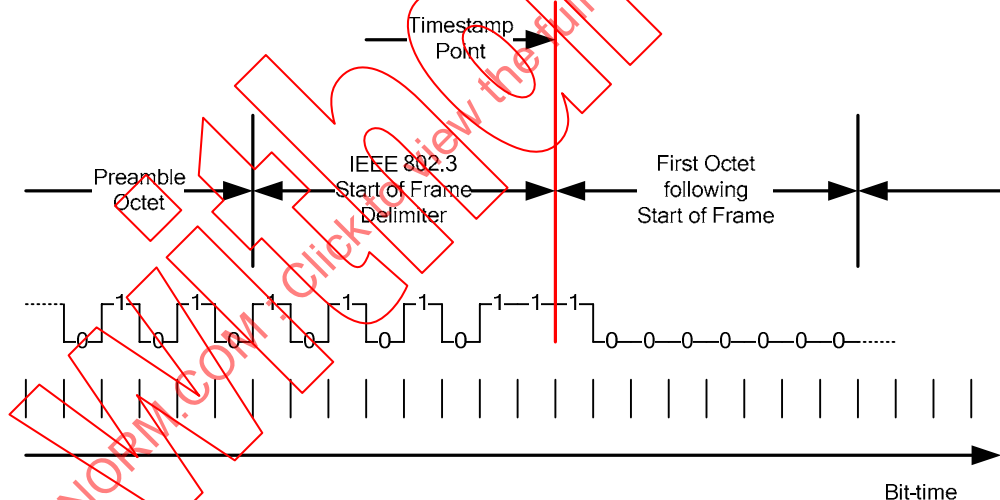
**Figure 14 – Message timestamp point**

Figure 15 shows the four timestamps used for synchronization.

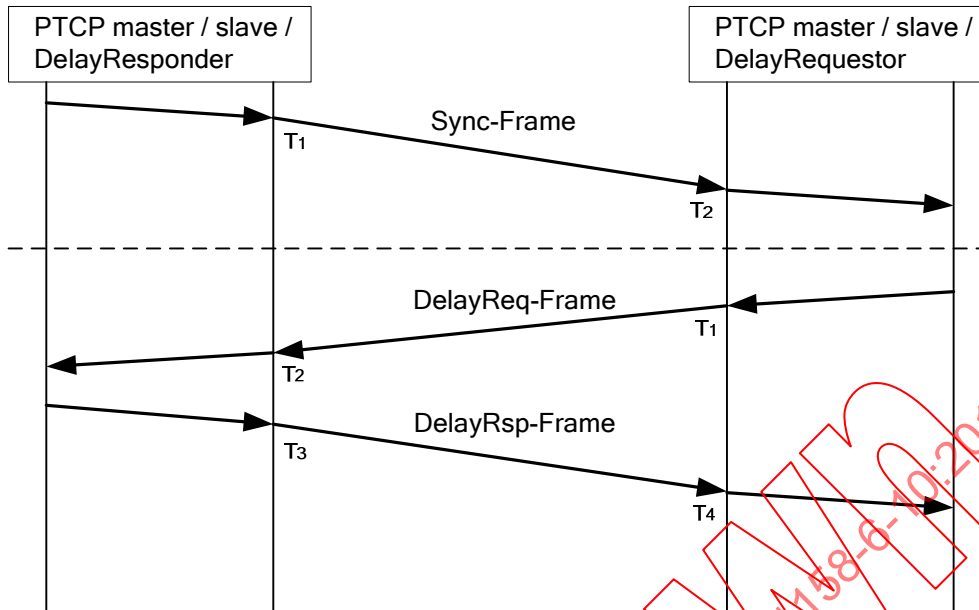


Figure 15 – Four message timestamps

SendTimeStamp T_1

is measured by the local clock of the port which sends a Sync- or a DelayReq-Frame.

ReceiptTimeStamp T_2

is measured by the local clock of the port which receives a Sync- or a DelayReq-Frame.

SendTimeStamp T_3

is measured by the local clock of the port which sends a DelayRes-Frame.

ReceiptTimeStamp T_4

is measured by the local clock of the port which receives a DelayRes-Frame.

4.4.4.1.2 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 16. 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.

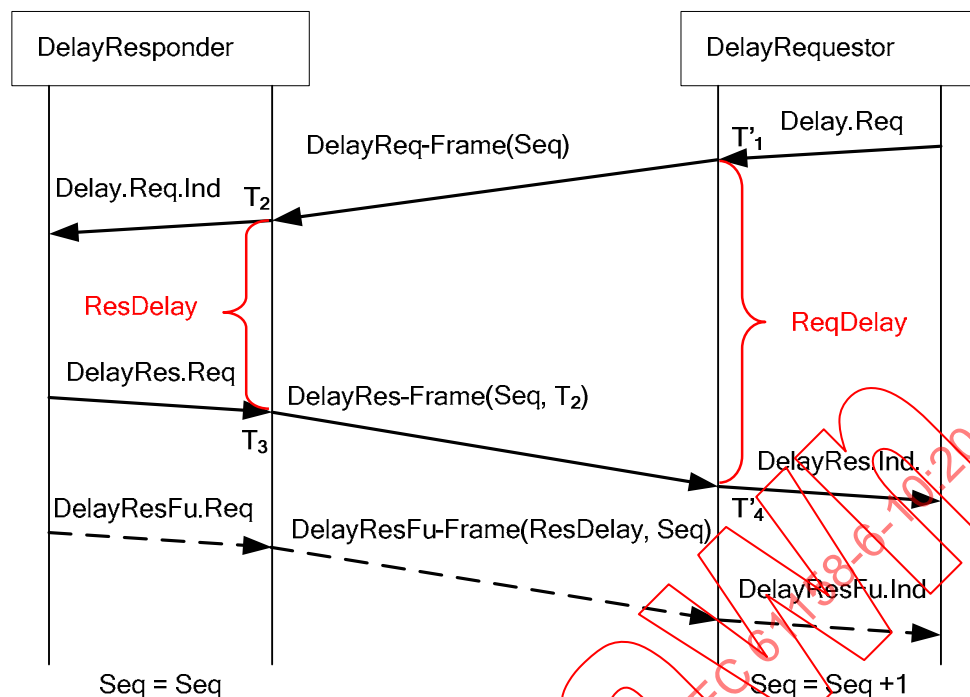


Figure 16 – Line delay protocol with follow up

If a time stamping unit is capable to calculate the response delay time on the fly and insert the response time in the delay response frame no delay follow up message is necessary.

The line delay measurement without DelayFuRes-message between DelayRequestor and DelayResponder is described in Figure 17.

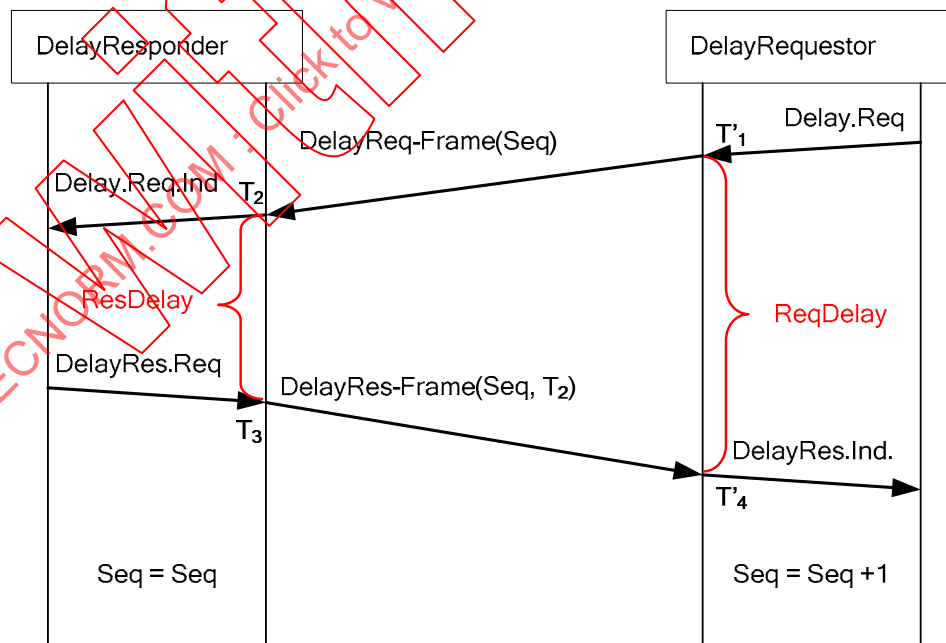


Figure 17 – Line delay protocol without follow up

The calculation of line delay and peer rate compensation factor is shown in following equations. If the DelayRes-PDU contains the field T_2 , the Equation (15) shall be used, otherwise Equation (16) shall be used.

$$\text{ResDelay} = T_3 - T_2 \quad (13)$$

$$\text{ReqDelay} = T'_4 - T'_1 \quad (14)$$

$$\text{RCF}_{\text{peer}} = \frac{T_2(\text{Seq}) - T_2(\text{Seq} - 1)}{T'_1(\text{Seq}) - T'_1(\text{Seq} - 1)} \quad (15)$$

$$\text{RCF}_{\text{peer}} = 1 \quad (16)$$

$$\text{ResDelay}_{\text{peer}} = \frac{\text{ResDelay}}{\text{RCF}_{\text{peer}}} \quad (17)$$

$$\text{LineDelay}_{\text{peer}} = \frac{\text{ReqDelay} - \text{ResDelay}_{\text{peer}}}{2} \quad (18)$$

$$\text{RCF}_{\text{Scal}} = (\text{RCF}_{\text{peer}} - 1) \times 2^{30} \quad (19)$$

$$\text{RCF}_{\text{Trunc}} = \text{Truncate\_to\_Integer32}(\text{RCF}_{\text{Scal}}) \quad (20)$$

$$\text{ScaledFrequencyOffset} = \text{RCF}_{\text{Trunc}} \quad (21)$$

The delay measurement shall be repeated in cycles.

4.4.4.1.3 Line delay calculation

The line delay between two devices supporting PTCP is determined as described in Figure 18.

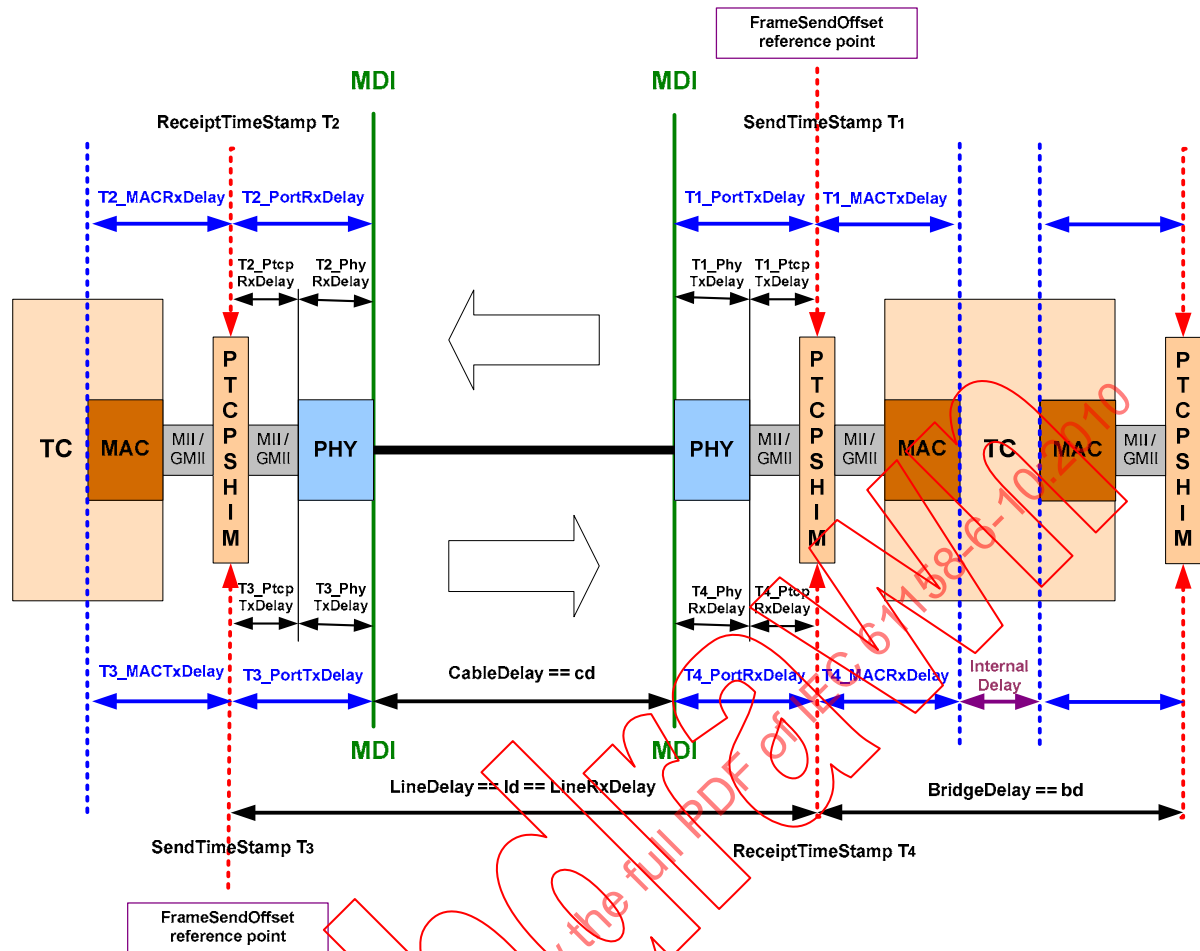


Figure 18 – Line delay measurement

NOTE 1 The Reduced Media Independent Interface (RMII) synchronises its internal 50 MHz clock with the 25 MHz clock of the PHY. It results in a quantization error of 20 ns. This jitter can not 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 quantization error of 45 ns. This jitter can not 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.

$$T1\_PortTxDelay = T1\_PhyTxDelay + T1\_PtcpTxDelay \quad (22)$$

$$T3\_PortTxDelay = T3\_PhyTxDelay + T3\_PtcpTxDelay \quad (23)$$

$$T2\_PortRxDelay = T2\_PhyRxDelay + T2\_PtcpRxDelay \quad (24)$$

$$T4\_PortRxDelay = T4\_PhyRxDelay + T4\_PtcpRxDelay \quad (25)$$

$$CableDelay = (ReqDelay - ResDelay - T1\_PortTxDelay - T2\_PortRxDelay - T3\_PortTxDelay - T4\_PortRxDelay) / 2 \quad (26)$$

$$LineDelay = CableDelay + T3\_PortTxDelay + T4\_PortRxDelay \quad (27)$$

LineSyncDelay is the line delay of the Sync-Frame. Due to the asymmetry of the involved communication path and due to the resolution of the time stamps it is basically impossible to determine the line delay exactly.

A PTCP clock can detect non PTCP neighbors by checking the line delay between nodes. As a switch delay is in the range of several (>2) μs , the delay between two nodes connected with 100 m Twisted Pair cabling (100 Base TX) is about 500 ns.

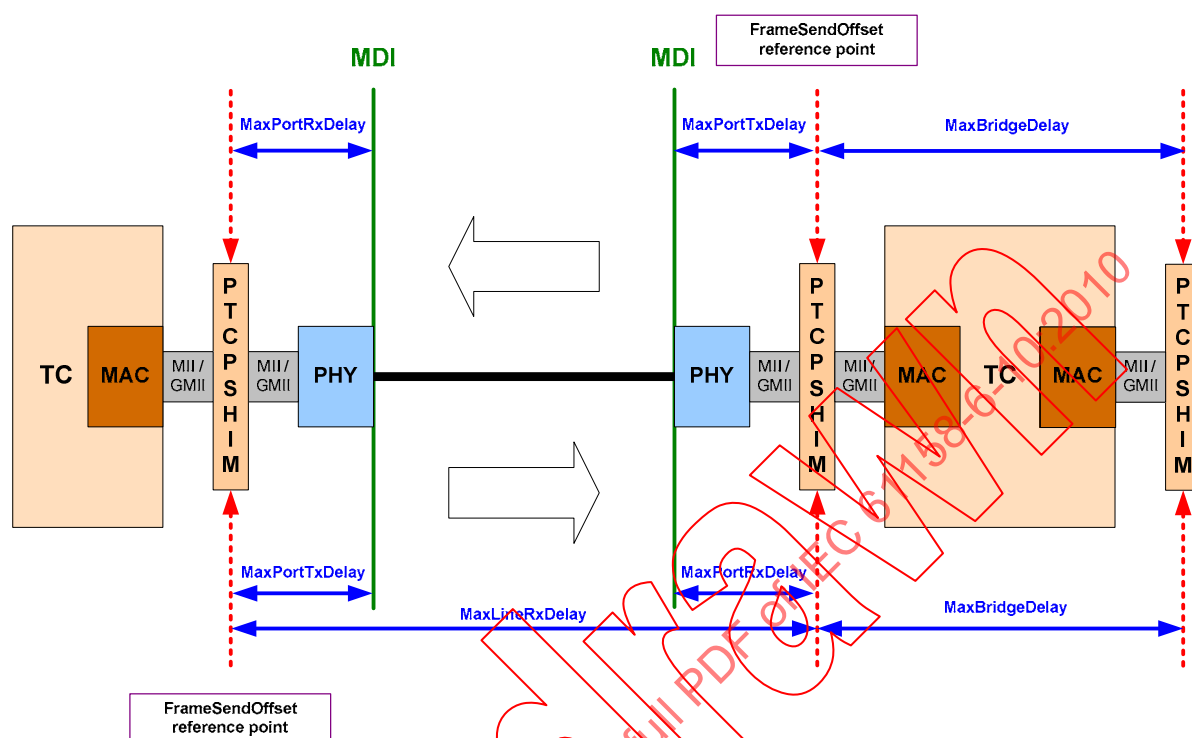


Figure 19 – Model parameter for GSDML usage

Figure 19 shows the model parameter for GSDML usage. The parameter MaxBridgeDelay, MaxPortTXDelay and MaxPortRXDelay offers engineering tools a calculation basis for RT_CLASS_3.

4.4.4.1.4 Sync-Frame forwarding

A device which forwards PTCP sync messages measures the bridge delay as shown in Figure 20 and adds this time to the delay field in the sync or follow up message.

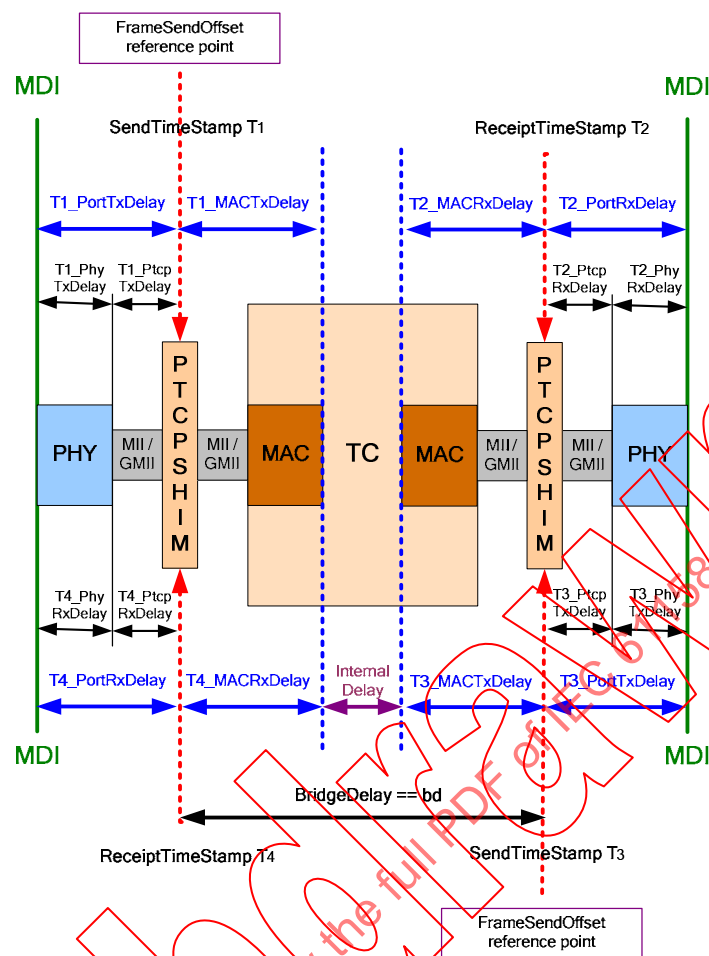


Figure 20 – Bridge delay measurement

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

$$\text{BridgeDelay} = T_3 - T_4 \quad (28)$$

$$\text{BridgeDelay}_{\text{SyncMaster}} = \text{BridgeDelay} \times \text{RCF}_{\text{SyncMaster}} \quad (29)$$

The principle of synchronization through a sequence of devices can be seen in Figure 21. 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.

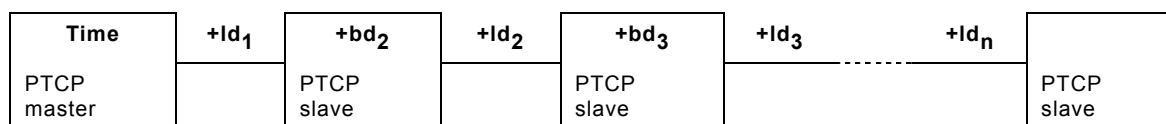


Figure 21 – Delay accumulation

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

adding its bridge delay bd_x (bridge delay measurement is shown in Figure 20)

adding the line delay ld_x (line delay measurement is shown in Figure 18),

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.

4.4.4.1.5 Rate compensation

4.4.4.1.5.1 Bridge delay

If the bridge delay is significant (several 100 μ s), then the values shall be corrected by rate compensation in order to achieve high precision synchronization.

EXAMPLE If a frame of 12 000 Bit times will be send out the delay of a subsequent time frame is about 120 μ s at fast Ethernet speed. If there is a clock drift of 100 PPM and the PTCP master has a clock drift of -100 PPM this would result in an error of 24 ns. If there is no priority for PTCP frames and there are 10 frames queued, this could result in an additional error of 240 ns which may be unacceptable. That means that a large delay value means less precision of the delay value. Each clock sampling at a node is always associated with a synchronization jitter. In addition to this, there is always control inaccuracy for each bridge. The maximum time difference between any two nodes of a network is dependent on the topology – cascading of bridges result in an inaccuracy depending on the level of cascading.

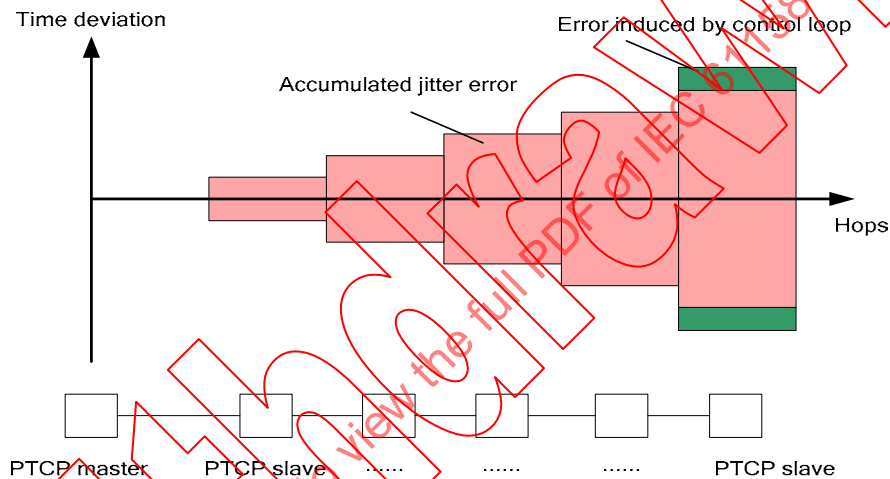


Figure 22 – Worst case accumulated time deviation of synchronization

The highest required accuracy for synchronization applies only to two adjacent nodes.

4.4.4.1.5.2 Line delay measurement

If the ReqDelay shown in Figure 17 is significant (several 100 μ s), then the values of ReqDelay and ResDelay shall be corrected by rate compensation in order to achieve high precision synchronization.

4.4.4.1.6 Time deviation measurement

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

Figure 23 and Figure 24 shows in principle the measurement of the deviation.

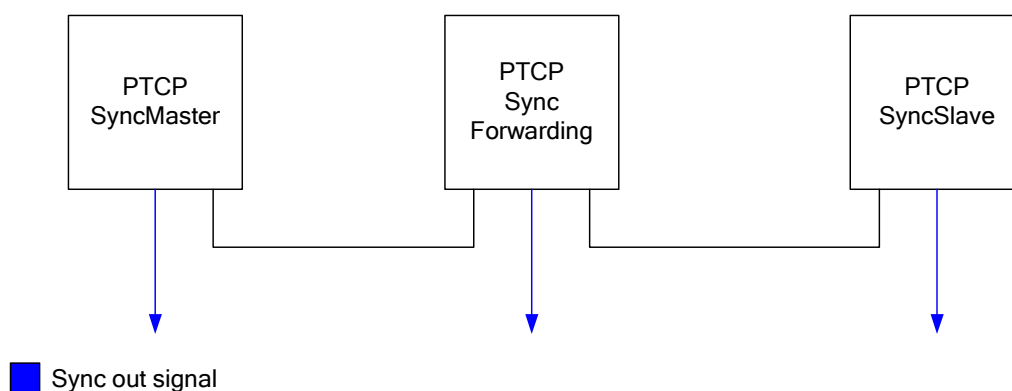


Figure 23 – Scheme for measurement of deviation

Every cycle as shown in Figure 13 shall be signaled for clock synchronization. Every second shall be signaled for time synchronization.

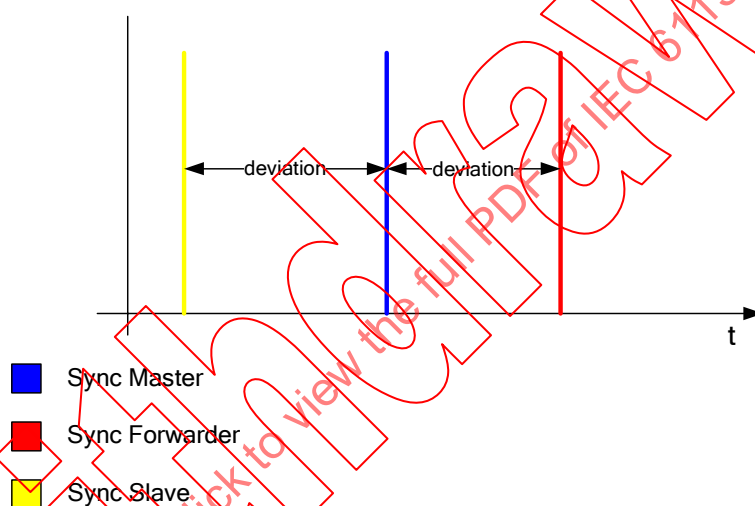


Figure 24 – Measurement of deviation

4.4.4.1.7 Synchronization Protocol

4.4.4.1.7.1 Synchronization with Sync-Frame

PTCP masters are sending Sync-Frames periodically. The synchronization scheme as described in Figure 25 requires transferring the exact sending time of the Sync-Frame as initial value of the delay.

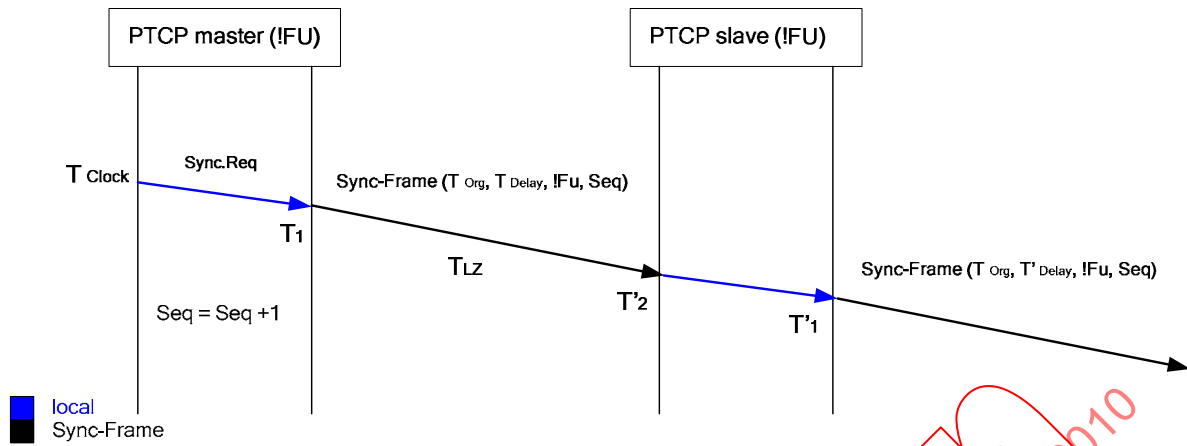


Figure 25 – Sending Sync-Frame without Follow Up-Frame

$$T_{Org} = T_{Clock} \quad (30)$$

$$T_{Delay} = T_1 - T_{Clock} \quad (31)$$

$$T'_{err} = T'_2 - (T_{Org} + T_{Delay} + T_{LZ}) \quad (32)$$

$$T'_{Delay} = T_{Delay} + T_{LZ} + (T'_1 - T'_2) \quad (33)$$

4.4.4.1.7.2 Synchronization with Sync- and FollowUp-Frame

Figure 26 shows a PTCP master which can not modify a frame before transmission. A PTCP master who is not able to put the exact sending time in a Sync-Frame will use a FollowUp-Frame.

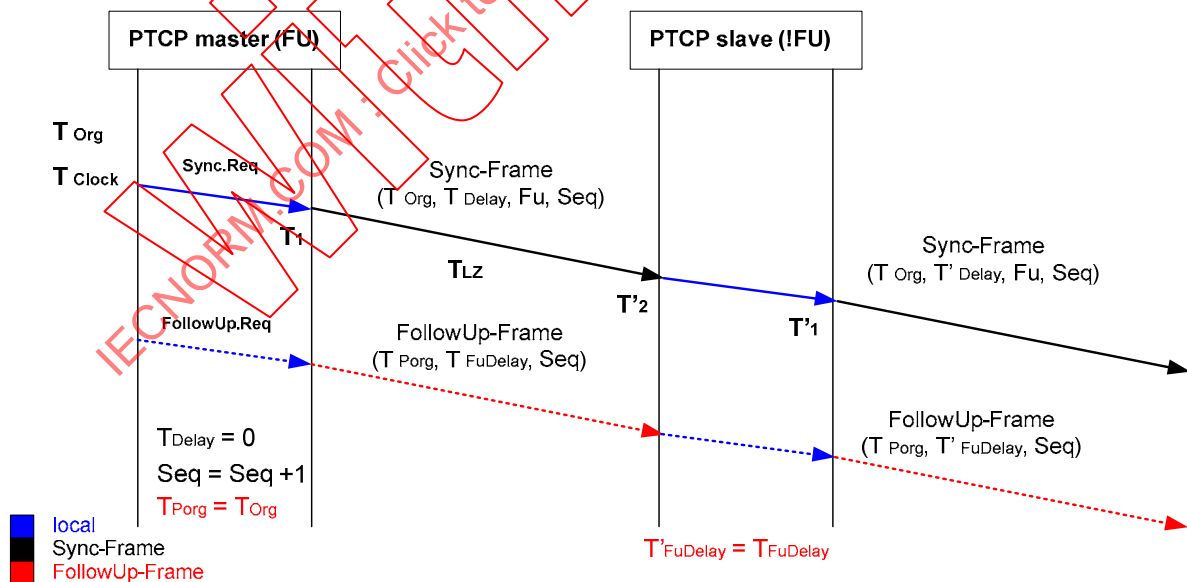


Figure 26 – Sending Sync-Frame with FollowUp-Frame

The delay field in the Sync-Frame shall be set to 0. The delay ($T_{FuDelay}$) in the associated FollowUp-Frame contains the initial value of the delay.

$$T_{Org} = T_{Clock} \quad (34)$$

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

$$T_{err} = T'_2 - (T_{Porg} + T'_{Delay} + T'_{FuDelay} + T'_{LZ}) \quad (36)$$

$$T'_{Delay} = T_{Delay} + T_{LZ} + (T'_1 - T'_2) \quad (37)$$

A PTCP slave as shown in Figure 27 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. A FollowUp-Frame is forwarded without modifications.

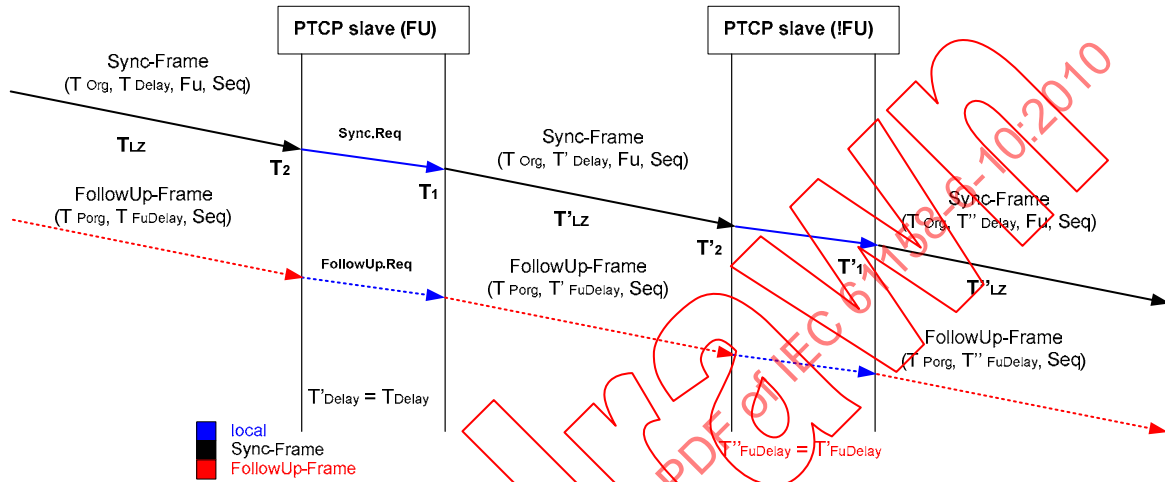


Figure 27 – Forwarding Sync- and FollowUp-Frame

$$T_{err} = T_2 - (T_{Porg} + T_{Delay} + T_{FuDelay} + T_{LZ}) \quad (38)$$

$$T'_{FuDelay} = T_{FuDelay} + T_{LZ} + (T_1 - T_2) \quad (39)$$

$$T'_{err} = T'_2 - (T_{Porg} + T'_{Delay} + T'_{FuDelay} + T'_{LZ}) \quad (40)$$

$$T''_{Delay} = T'_{Delay} + T'_{LZ} + (T'_1 - T'_2) \quad (41)$$

4.4.4.1.7.3 Concurrent operation of synchronization variants

Some implementations of IEEE 802.3 do not allow modifying a frame during transmission. The exact delay between T_{Org} and the time sending Sync-Frame will be sent in a second frame which is called FollowUp-Frame. A PTCP slave (FU-Node), which is not capable of adding its internal bridge delay and the line delay to the delay value in the Sync-Frame as it passes through, puts its bridge delay and the line delay to the delay value in a generated FollowUp-Frame. This principle is shown in Figure 28.

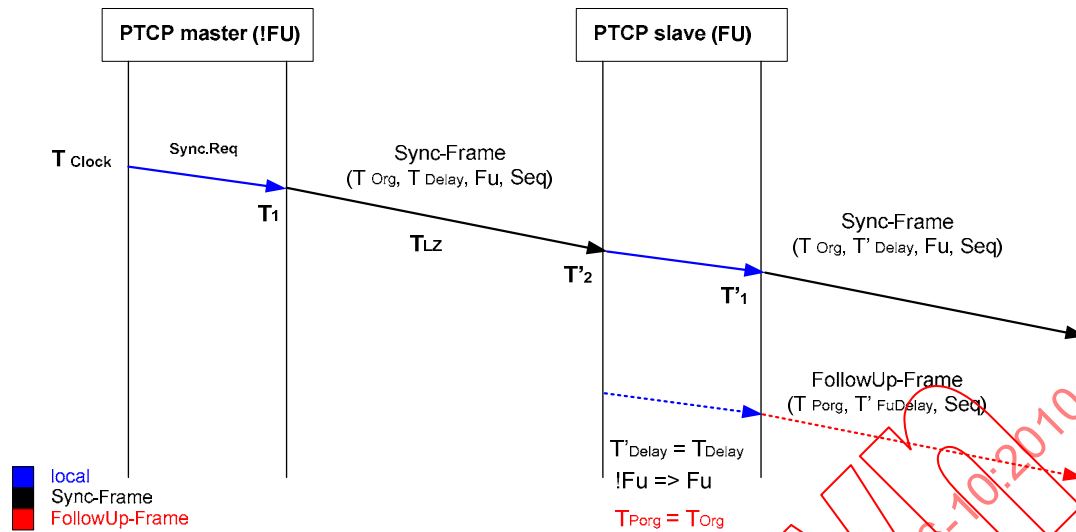


Figure 28 – Transition between Synchronization Variants

$$T_{Org} = T_{Clock} \quad (42)$$

$$T_{Offset} = T_2 - (T_{Org} + T_{Delay} + T_{LZ}) \quad (43)$$

$$T'_{FuDelay} = T_{LZ} + (T_1 - T_2) \quad (44)$$

$$T'_{err} = T'_2 - (T_{Porg} + T'_{Delay} + T'_{FuDelay} + T'_{LZ}) \quad (45)$$

$$T''_{Delay} = T'_{Delay} + T'_{LZ} + (T'_1 - T'_2) \quad (46)$$

4.4.4.1.8 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 BMA algorithm will cancel the activities of a Sync master.

The line delay values can be restricted for some technologies (e.g. 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 (DelayRequest)

4.4.4.2.1.1 Primitive definitions

4.4.4.2.1.1.1 Primitives exchanged between DelayRequest and ASE

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

4.4.4.2.1.2 State machine description

The line delay measurement shall be initiated by each port and should be repeated in intervals of 8 seconds. The state machine which sends a delay request frame is called DelayRequestor.

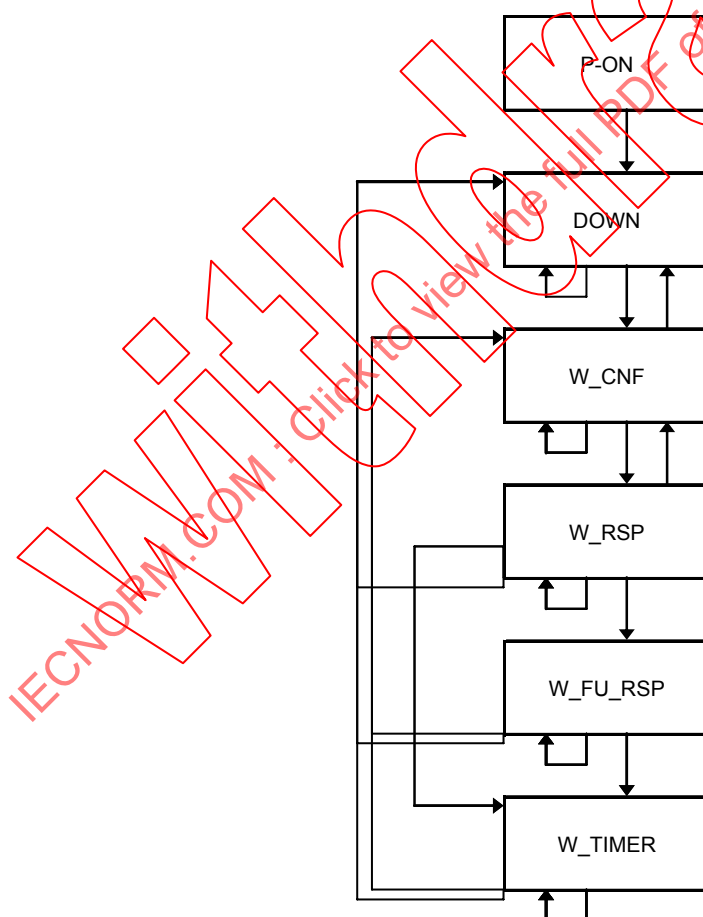


Figure 29 – State transition diagram of delay request

States of the **DelayRequest**

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.

Local variables of the **DelayRequest**

Tx_TStamp_T1

This local variable contains the transmit timestamp of the delay request frame.

Rx_TStamp_T4

This local variable contains the receive timestamp of the delay response frame.

Rx_TStamp1_T2

This local variable contains a receive timestamp of the delay request frame by the delay responder.

Rx_TStamp2_T2

This local variable contains a receive timestamp of the delay request frame by the delay responder.

SequenceID

This local variable contains the sequence number of the delay request message. It shall be incremented with every delay measurement.

4.4.4.2.1.3 DelayRequest state table

Table 97 contains the state table used by DelayRequest.

Table 97 – DelayRequest state table

#	Current State	Event /Condition =>Action	Next State
1	P-ON	=> NRepeat := 0 SequenceID := 0 RESET_RCF_PEER RESET_LINE_DELAY	DOWN
2	DOWN	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) /LINK_OK => ignore	DOWN
3	DOWN	MAUType_Change.ind (+) (PortID, MAU_Type, LINK_Status) /LINK_OK => DelayT.start (DELAY_REQ_SHORT_INTERVAL) DelayReq_Req (Port, PTCP_DELAY_REQ_PDU)	W_CNF
4	W_CNF	DelayReq_Cnf (-) (D_Port, TStamp, Status) => SequenceID++	W_TIMER
5	W_CNF	DelayReq_Cnf (+) (D_Port, TStamp, Status) / Status == OK => Tx_Tstamp_T1 := TStamp	W_RSP
6	W_CNF	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) /LINK_OK => NRepeat := 0 DelayT.stop SequenceID++ RESET_RCF_PEER RESET_LINE_DELAY	DOWN
7	W_CNF	MAUType_Change.ind (+) (PortID, MAU_Type, LINK_Status) /LINK_OK => ignore	W_CNF
8	W_CNF	DelayT.expired => SequenceID++ DelayT.start (DELAY_REQ_SHORT_INTERVAL) DelayReq_Req (Port, PTCP_DELAY_REQ_PDU)	W_CNF
9	W_CNF	LineT.expired => RESET_LINE_DELAY	W_CNF
10	W_RSP	DelayResp_Ind (-) (S_Port, TStamp, PTCP_DELAY_RES_PDU) => ignore	W_RSP
11	W_RSP	DelayResp_Ind (+) (S_Port, TStamp, PTCP_DELAY_RES_PDU) /CHECK_DELAY_RSP_VALID && CHECK_DELAY_RSP_WITH_FU_RSP => Rx_Tstamp2_T2 := Rx_Tstamp1_T2 Rx_Tstamp1_T2 := PTCP_DELAY_RES_PDU.T2 TimeStampDelay := PTCP_DELAY_RES_PDU.Delay Rx_Tstamp_T4 := Tstamp if (NRepeat > 0) CALC_RCF_PEER	W_FU_RSP
12	W_RSP	DelayResp_Ind (+) (S_Port, TStamp, PTCP_DELAY_RES_PDU) /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	W_TIMER

#	Current State	Event /Condition =>Action	Next State
		if (NRepeat > 0) CALC_RCF_PEER CALC_LINE_DELAY LineT.start (TIMEOUT_LINE_DELAY) NRepeat++	
13	W_RSP	DelayResp_Ind (+) (S_Port, TStamp, PTCP_DELAY_RES_PDU) /CHECK_DELAY_RSP_VALID && !CHECK_DELAY_RSP_WITH_FU_RSP && !CHECK_DELAY_VALID => SET_DUMMY_LINE_DELAY LineT.start (TIMEOUT_LINE_DELAY) DelayT.start (DELAY_REQ_SHORT_INTERVAL)	W_TIMER
14	W_RSP	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) /LINK_OK => NRepeat := 0 DelayT.stop SequenceID++ RESET_RCF_PEER RESET_LINE_DELAY	DOWN
15	W_RSP	MAUType_Change.ind (+) (PortID, MAU_Type, LINK_Status) /LINK_OK => ignore	W_RSP
16	W_RSP	DelayT.expired => SequenceID++ Nrepeat := 0 RESET_RCF_PEER DelayT.start (DELAY_REQ_LONG_INTERVAL)	W_TIMER
17	W_RSP	LineT.expired => RESET_LINE_DELAY	W_RSP
18	W_FU_RSP	DelayResp_Ind (-) (S_Port, TStamp, PTCP_DELAY_RES_PDU) => ignore	W_FU_RSP
19	W_FU_RSP	DelayResp_Ind (+) (S_Port, TStamp, PTCP_DELAY_RES_PDU) /CHECK_DELAY_RSP_VALID => NRepeat := 0 DelayT.stop DelayT.start (DELAY_REQ_LONG_INTERVAL) RESET_LINE_DELAY	W_TIMER
20	W_FU_RSP	DelayFuResp_Ind (-) (S_Port, TStamp, PTCP_DELAY_FURES_PDU) => ignore	W_FU_RSP
21	W_FU_RSP	DelayFuResp_Ind (+) (S_Port, TStamp, PTCP_DELAY_RES_PDU) /CHECK_DELAY_FU_RSP_VALID => Delay := Delay + PTCP_DELAY_FURES_PDU.Delay if (NRepeat > 0) CALC_LINE_DELAY LineT.start (TIMEOUT_LINE_DELAY) NRepeat++	W_TIMER
22	W_FU_RSP	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) /LINK_OK => NRepeat := 0 DelayT.stop SequenceID++ RESET_LINE_DELAY	DOWN
23	W_FU_RSP	MAUType_Change.ind (+) (PortID, MAU_Type, LINK_Status) /LINK_OK => ignore	W_FU_RSP

#	Current State	Event /Condition =>Action	Next State
24	W_FU_RSP	DelayT.expired => SequenceID++ DelayT.start (DELAY_REQ_SHORT_INTERVAL) DelayReq_Req (Port, PTCP_DELAY_REQ_PDU)	W_CNF
25	W_FU_RSP	LineT.expired => RESET_LINE_DELAY	W_FU_RSP
26	W_TIMER	DelayResp_Ind (-) (S_Port, TStamp, PTCP_DELAY_RES_PDU) => ignore	W_TIMER
27	W_TIMER	DelayResp_Ind (+) (S_Port, TStamp, PTCP_DELAY_RES_PDU) /CHECK_DELAY_RSP_VALID => NRepeat := 0 RESET_RCF_PEER RESET_LINE_DELAY DelayT.stop DelayT.start (DELAY_REQ_LONG_INTERVAL)	W_TIMER
28	W_TIMER	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) /!LINK_OK => NRepeat := 0 SequenceID++ DelayT.stop RESET_RCF_PEER RESET_LINE_DELAY	DOWN
29	W_TIMER	MAUType_Change.ind (+) (PortID, MAU_Type, LINK_Status) /LINK_OK => ignore	W_TIMER
30	W_TIMER	DelayT.expired /NRepeat < MAX_DELAY_REQ_REPEAT => SequenceID++ DelayT.start (DELAY_REQ_SHORT_INTERVAL) DelayReq_Req (Port, PTCP_DELAY_REQ_PDU)	W_CNF
31	W_TIMER	DelayT.expired /NRepeat >= MAX_DELAY_REQ_REPEAT => NRepeat := 0 RESET_RCF_PEER DelayT.start (DELAY_REQ_LONG_INTERVAL)	W_TIMER
32	W_TIMER	LineT.expired => RESET_LINE_DELAY	W_TIMER

4.4.4.2.1.4 Macros

The Table 98 contains the macros used by DelayRequest.

Table 98 – Macros used by DelayRequest

Name	Meaning
CHECK_DELAY_RSP_VALID	Check parameter of PTCP_DelayResPDU PTCP_Sequence == SequenceID PTCP_RequestSourceAddress == Local_SA
CHECK_DELAY_RSP_WITH_FU_RSP	Check FrameID == 0xFF41 of PTCP_DelayResPDU
CHECK_DELAY_FU_RSP_VALID	Check parameter of PTCP_DelayFuResPDU PTCP_Sequence == SequenceID PTCP_PortMACAddress == Local_SA
CHECK_DELAY_RSP_HIGH_ACCURACY	Check parameter for high accuracy of PTCP_DelayResPDU PTCP_SubdomainUUID := Subdomain UUID of PTCP_DelayReqPDU PTCP_MasterSourceAddress == Sync master source MAC of PTCP_DelayReqPDU PTCP_SyncID == SyncID of PTCP_DelayReqPDU
CALC_LINE_DELAY	Line delay is the arithmetical average value of the last 8 delay measurements
RESET_LINE_DELAY	Set line delay to zero
SET_DUMMY_LINE_DELAY	Set line delay to one
CALC_RCF_PEER	Calculation of the frequency deviation to the neighbour
RESET_RCF_PEER	Set RCF_PEER to one

4.4.4.2.1.5 Functions

The Table 99 contains the functions used by DelayRequest.

Table 99 – Functions used by DelayRequest

Name	Meaning
DelayReq_Req (Port, PTCP_DELAY_REQ_PDU)	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)
DelayReq_Cnf (D_Port, TStamp, Status)	LMPM_P_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status
DelayResp_Ind (S_Port, TStamp, PTCP_DELAY_RSP_PDU)	Receive PTCP-PDU according PTCP_DELAY_RSP_PDU LMPM_P_Data.ind (CREP, S_Port, TSamp, DA, SA, A_SDU) Assignments: PTCP_DELAY_RSP_PDU := A_SDU without LT and FrameID
DelayFuResp_Ind (S_Port, TStamp, PTCP_DELAY_FU_RSP_PDU)	Receive PTCP-PDU according PTCP_DELAY_FU_RSP_PDU LMPM_P_Data.ind (CREP, D_Port, TSamp, DA, SA, A_SDU) Assignments: PTCP_DELAY_FU_RSP_PDU := A_SDU without LT and FrameID

4.4.4.2.1.5.1 DelayReq_Req, DelayReq_Cnf, DelayResp_Ind and DelayFuResp_Ind

Local macros used to send PTCP_DelayReqPDU or to receive PTCP_DelayResPDU or PTCP_DelayFuResPDU.

4.4.4.2.2 Line Delay Response Protocol Machine (DelayResponse)

4.4.4.2.2.1 State machine description

Each delay request message shall be immediately responded with a delay response messages. The state machine which receives a delay request frame is called DelayResponder.

The following Figure 32 shows the delay response state machine:

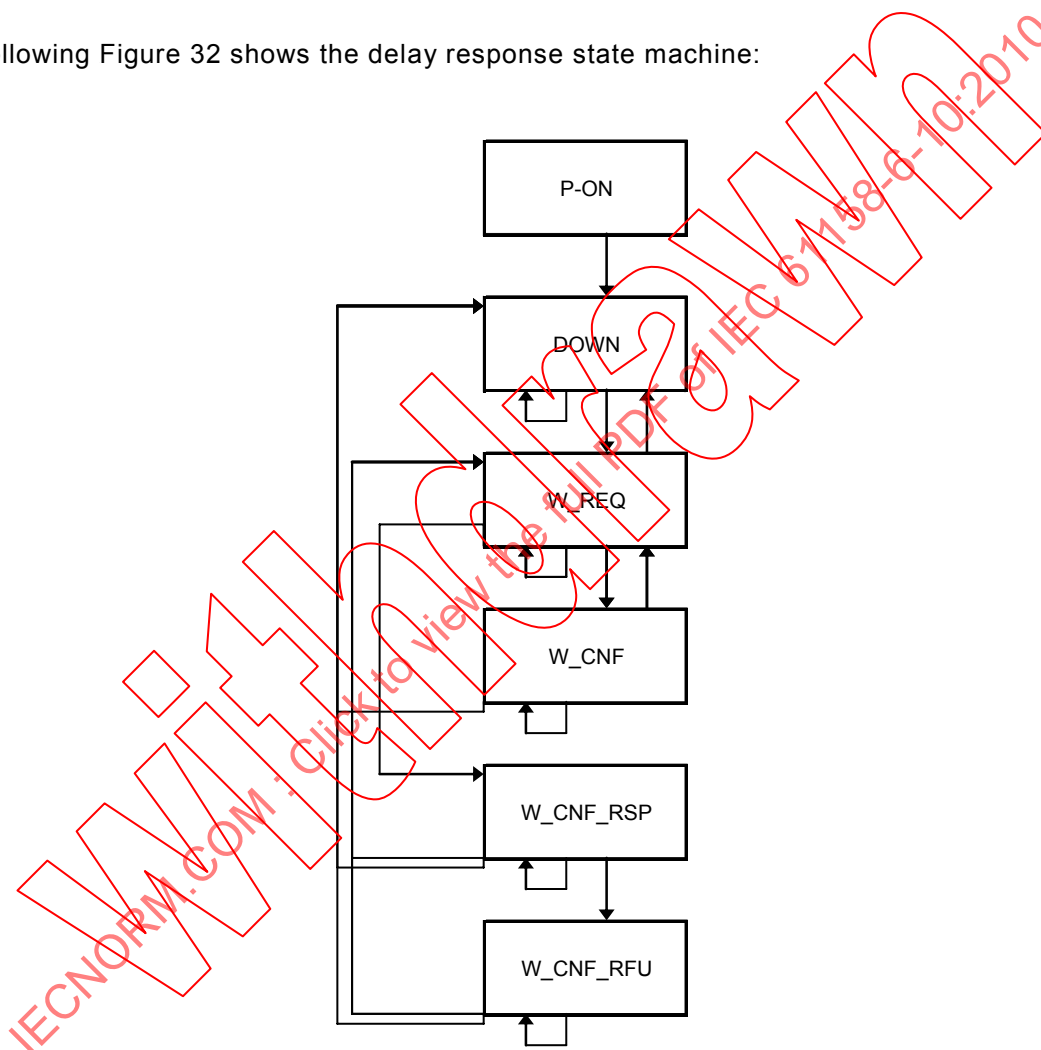


Figure 30 – State transition diagram of delay response

States of the DelayResponse

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.

Local variables of the DelayResponse

Rx_TStamp_T2

This local variable contains the receive time stamp of the delay request message.

Tx_TStamp_T3

This local variable contains the transmit time stamp of the delay response message.

4.4.4.2.2.2 DelayResponse state table

Table 100 contains the state table used by DelayResponse.

Withdrawing
IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 100 – DelayResponse state table

#	Current State	Event /Condition =>Action	Next State
1	P-ON	=>	DOWN
2	DOWN	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) => ignore	DOWN
3	DOWN	MAUType_Change.ind (+) (PortID, MAU_Type, LINK_Status) /LINK_Status == Up && MAU_Type == FULL_DUPLEX =>	W_REQ
4	W_REQ	DelayReq_Ind (-) (S_Port, TStamp, PTCP_DELAY_REQ_PDU) => ignore	W_REQ
5	W_REQ	DelayReq_Ind (+) (S_Port, TStamp, PTCP_DELAY_REQ_PDU) /FU_RES => store PTCP_DELAY_REQ_PDU Rx_TStamp_T2 := TStamp PTCP_DEALY_RES_PDU.T2TimeStamp := Rx_Tstamp_T2 DelayResp_Req (Port, PTCP_DELAY_RES_PDU)	W_CNF_RSP
6	W_REQ	DelayReq_Ind (+) (S_Port, TStamp, PTCP_DELAY_REQ_PDU) /!FU_RES => store PTCP_DELAY_REQ_PDU Rx_TStamp_T2 := TStamp PTCP_DEALY_RES_PDU.T2TimeStamp := Rx_Tstamp_T2 DelayResp_Req (Port, PTCP_DELAY_RES_PDU)	W_CNF
7	W_REQ	DelayReq_Ind (+) (S_Port, TStamp, PTCP_DELAY_REQ_PDU) /!FU_TIMESTAMP => store PTCP_DELAY_REQ_PDU PTCP_DEALY_RES_PDU.Delay := 0 DelayResp_Req (Port, PTCP_DELAY_RES_PDU)	W_CNF
8	W_REQ	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) =>	DOWN
9	W_REQ	MAUType_Change.ind (+) (PortID, MAU_Type, LINK_Status) /LINK_Status == Up && MAU_Type == FULL_DUPLEX => ignore	W_REQ
10	W_CNF	DelayResp_Cnf (-) (D_Port, TStamp, Status) =>	W_REQ
11	W_CNF	DelayResp_Cnf (+) (D_Port, TStamp, Status) /Status == OK =>	W_REQ
12	W_CNF	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) =>	DOWN
13	W_CNF	MAUType_Change.ind (+) (PortID, MAU_Type, LINK_Status) /LINK_Status == Up && MAU_Type == FULL_DUPLEX=>ignore	W_CNF
14	W_CNF_RSP	DelayResp_Cnf (-) (D_Port, TStamp, Status) =>	W_REQ
15	W_CNF_RSP	DelayResp_Cnf (+) (D_Port, TStamp, Status) /Status == OK => Tx_TStamp_T3 := TStamp PTCP_DEALY_RES_PDU.Delay := CALC_RESIDENTIAL_TIME DelayFuResp_Req (Port, PTCP_DELAY_FU_RES_PDU)	W_CNF_RFU
16	W_CNF_RSP	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) =>	DOWN

#	Current State	Event /Condition =>Action	Next State
17	W_CNF_RSP	MAUType_Change.ind (+) (PortID, MAU_Type, LINK_Status) /LINK_Status == Up && MAU_Type == FULL_DUPLEX => ignore	W_CNF_RSP
18	W_CNF_RFU	DelayFuResp_Cnf (-) (D_Port, TStamp, Status) =>	W_REQ
19	W_CNF_RFU	DelayFuResp_Cnf (+) (D_Port, TStamp, Status) /Status == OK =>	W_REQ
20	W_CNF_RFU	MAUType_Change.ind (-) (PortID, MAU_Type, LINK_Status) =>	DOWN
21	W_CNF_RFU	MAUType_Change.ind (PortID, MAU_Type, LINK_Status) /LINK_Status == Up && MAU_Type == FULL_DUPLEX => ignore	W_CNF_RFU

4.4.4.2.2.3 Macros

The Table 101 contains the macros used by DelayResponse.

Table 101 – Macros used by DelayResponse

Name	Meaning
CALC_RESIDENTIAL_TIME	Calculate the residential time ResTime := (Tx_Stamp_T3 - Rx_Stamp_T2)

4.4.4.2.2.4 Functions

The Table 102 contains the functions used by the DelayResponse.

Table 102 – Functions used by DelayResponse

Name	Meaning
DelayResp_Req (Port, PTCP_DELAY_RSP_PDU)	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_RSP_PDU)	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)	LMPM_P_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status
DelayFuResp_Cnf (D_Port, TStamp, Status)	LMPM_P_Data.conf (CREP, D_Port, TStamp, LMPM_Status) Assignments: Status := LMPM_status
DelayReq_Ind (S_Port, TStamp, PTCP_DELAY_REQ_PDU)	Receive PTCP-PDU according PTCP_DELAY_REQ_PDU LMPM_P_Data.ind (CREP, S_Port, TSamp, DA, SA, A_SDU) Assignments: PTCP_DELAY_REQ_PDU := A_SDU without LT and FrameID

4.4.4.2.4.1 DelayResp_Req, DelayFuResp_Req, DelayResp_Cnf, DelayFuResp and DelayReq_Ind

Local macros used to send PTCP_DelayResPDU and PTCP_DelayFuResPDU or to receive PTCP_DelayReqPDU.

4.4.4.3 Overview for master slave state tables

Figure 31 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.

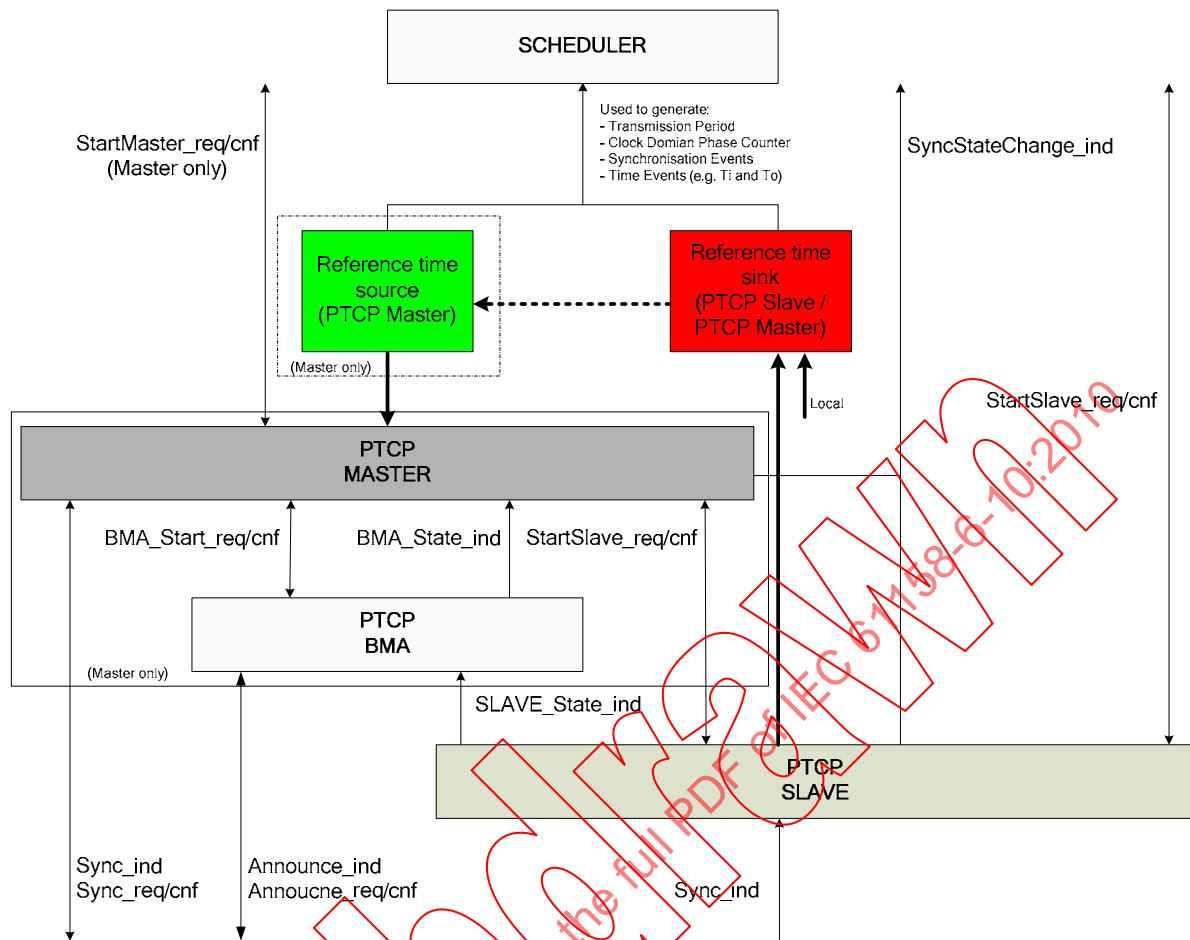


Figure 31 – Overview of PTCP

4.4.4.4 Best-Master-Algorithm Protocol Machine (BMA)

4.4.4.4.1 Primitive definitions

4.4.4.4.1.1 Primitives exchanged between BMA and ASE

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

4.4.4.4.2 State machine description

The BMA-state-machine shall elect the best master from all announced master-capable devices during the start up 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 behaviour 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 become master or secondary master will be decided during the start up period by the best master algorithm. Every potential master has to 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 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.

The following Figure 32 shows the BMA state machine:

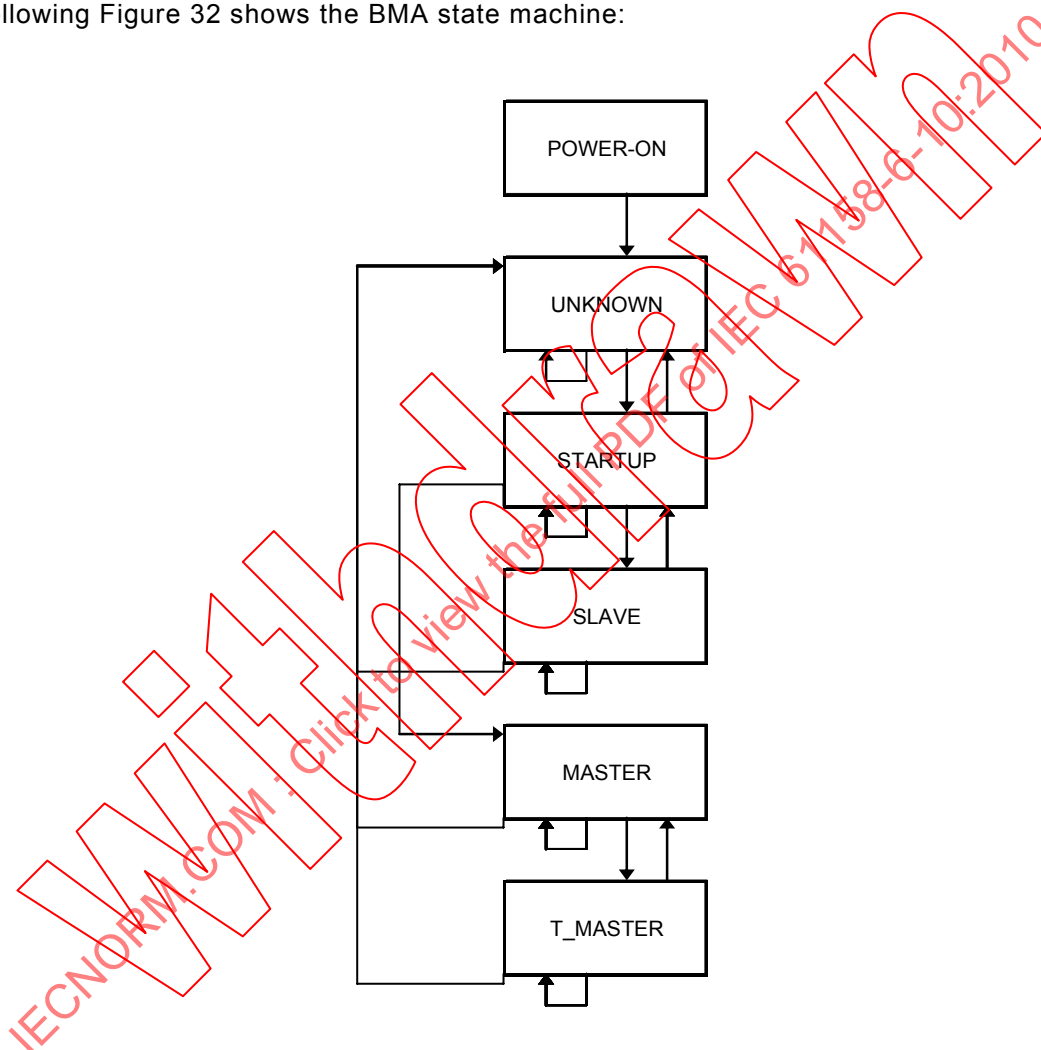


Figure 32 – State transition diagram of BMA

States of the BMA

POWER_ON

Data initialization

UNKNOWN

The synchronization role is unknown. A role shall 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 von announc messages from an other potential master in the sync domain.

4.4.4.4.3 BMA state table

Table 103, Table 104 and Table 105 contains the state tables used by BMA.

Withdrawing
IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 103 – BMA state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> ini MasterList SequenceID := 0 BestMaster := FALSE ACTIVE_MASTER_FLAG := FALSE AGING_MASTER_LIST MasterListAgingT.start (MASTER_AGING_TIME)	UNKNOWN
2	UNKNOWN	BMA_Start_req () => Status := OK AnnTimer.start (AnnInterval) StartupTimer.start (3xMasterStartupTime) BMA_Start_conf (Status) Announce_req (PTCP_ANNOUNCE_PDU)	STARTUP
3	UNKNOWN	BMA_Stop_req () => AnnTimer.stop () Status := OK BMA_Stop_conf (Status)	UNKNOWN
4	UNKNOWN	MasterListAgingT.expired () => AGING_MASTER_LIST MasterListAgingT.start (MASTER_AGING_TIME)	UNKNOWN
5	STARTUP	BMA_Start_req () => Status := WRONG_SEQUENCE BMA_Start_conf (Status)	STARTUP
6	STARTUP	BMA_Stop_req () => AnnTimer.stop () StartupTimer.stop () Status := OK BMA_Stop_conf (Status)	UNKNOWN
7	STARTUP	Announce_cnf (-) => Announce_req (PTCP_ANNOUNCE_PDU)	STARTUP
8	STARTUP	Announce_cnf (+) => ignore	STARTUP
9	STARTUP	SLAVE_State_ind (State) => ignore	STARTUP
10	STARTUP	Announce_ind (PTCP_ANNOUNCE_PDU) /!VALID_SYNC_MASTER => ignore	STARTUP
11	STARTUP	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && !CHECK_IN_MASTER_LIST => ADD_TO_MASTER_LIST	STARTUP
12	STARTUP	Announce_ind (PTCP_ANNOUNCE_PDU)/VALID_SYNC_MASTER&& CHECK_IN_MASTER_LIST&& !VALID_ANNOUNCE=>	STARTUP
13	STARTUP	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST && VALID_ANNOUNCE => UPDATE_MASTER_ENTRY	STARTUP

#	Current State	Event /Condition =>Action	Next State
14	STARTUP	AnnTimer.expired () /!LOCAL_IS_BEST_MASTER => BestMaster := FALSE StartupTimer.stop () AnnTimer.start (AnnInterval) BMA_State_ind (SLAVE)	SLAVE
15	STARTUP	AnnTimer.expired () /LOCAL_IS_BEST_MASTER => BestMaster := TRUE SequenceID++ AnnTimer.start (AnnInterval) BMA_State_ind (MASTER) Announce_req (PTCP_ANNOUNCE_PDU)	STARTUP
16	STARTUP	StartupTimer.expired () => ACTIVE_MASTER_FLAG := TRUE	MASTER
17	STARTUP	MasterListAgingT.expired () => AGING_MASTER_LIST MasterListAgingT.start (MASTER_AGING_TIME)	STARTUP
18	SLAVE	BMA_Start_req () => Status := WRONG_SEQUENCE BMA_Start_conf (Status)	SLAVE
19	SLAVE	BMA_Stop_req () => AnnTimer.stop () Status := OK BMS_Stop_conf (Status)	UNKNOWN
20	SLAVE	Announce_cnf () =>	SLAVE
21	SLAVE	SLAVE_State_ind (State) /State != MASTER_LOST =>	SLAVE
22	SLAVE	SLAVE_State_ind (State) /State == MASTER_LOST && !BestMaster =>	SLAVE
23	SLAVE	SLAVE_State_ind (State) /State == MASTER_LOST&& BestMaster=>SequenceID++StartupTimer.start (MasterStartupTime)BMA_State_ind (MASTER)Announce_req (PTCP_ANNOUNCE_PDU)	STARTUP
24	SLAVE	Announce_ind (PTCP_ANNOUNCE_PDU) /!VALID_SYNC_MASTER => ignore	SLAVE
25	SLAVE	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && !CHECK_IN_MASTER_LIST => ADD_TO_MASTER_LIST	SLAVE
26	SLAVE	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST && !VALID_ANNOUNCE =>	SLAVE

#	Current State	Event /Condition =>Action	Next State
27	SLAVE	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST && VALID_ANNOUNCE => UPDATE_MASTER_ENTRY	SLAVE
28	SLAVE	AnnTimer.expired () /!LOCAL_IS_BEST_MASTER => BestMaster := FALSE AnnTimer.start (AnnInterval)	SLAVE
29	SLAVE	AnnTimer.expired () /LOCAL_IS_BEST_MASTER => BestMaster := TRUE AnnTimer.start (AnnInterval)	SLAVE
30	SLAVE	MasterListAgingT.expired () => AGING_MASTER_LIST MasterListAgingT.start (MASTER_AGING_TIME)	SLAVE
31	MASTER	BMA_Start_req () => Status := WRONG_SEQUENCE BMA_Start_conf (Status)	MASTER
32	MASTER	BMA_Stop_req () => AnnTimer.stop () Status := OK BMS_Stop_conf (Status)	UNKNOWN
33	MASTER	Announce_cnf () =>	MASTER
34	MASTER	SLAVE_State_ind (State) => ignore	MASTER
35	MASTER	Announce_ind (PTCP_ANNOUNCE_PDU) /!VALID_SYNC_MASTER => ignore	MASTER
36	MASTER	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && !CHECK_IN_MASTER_LIST => ADD_TO_MASTER_LIST	MASTER
37	MASTER	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST && !VALID_ANNOUNCE =>	MASTER
38	MASTER	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST && VALID_ANNOUNCE => UPDATE_MASTER_ENTRY	MASTER
39	MASTER	AnnTimer.expired () /!MASTER_LIST_EMPTY => SequenceID++ AnnTimer.start (AnnInterval) StartupTimer.start (MasterStartupTime) Announce_req (PTCP_ANNOUNCE_PDU)	T_MASTER

#	Current State	Event /Condition =>Action	Next State
40	MASTER	AnnTimer.expired () /MASTER_LIST_EMPTY => AnnTimer.start (AnnInterval)	MASTER
41	MASTER	MasterListAgingT.expired () => AGING_MASTER_LIST MasterListAgingT.start (MASTER_AGING_TIME)	MASTER
42	T_MASTER	BMA_Start_req () => Status := WRONG_SEQUENCE BMA_Start_conf (Status)	T_MASTER
43	T_MASTER	BMA_Stop_req () => AnnTimer.stop () Status := OK BMS_Stop_conf (Status)	UNKNOWN
44	T_MASTER	Announce_cnf (-) =>Announce_req (PTCP_ANNOUNCE_PDU)	T_MASTER
45	T_MASTER	Announce_cnf (+) => ignore	T_MASTER
46	T_MASTER	SLAVE_State_ind (State) => ignore	T_MASTER
47	T_MASTER	Announce_ind (PTCP_ANNOUNCE_PDU) /!VALID_SYNC_MASTER => ignore	T_MASTER
48	T_MASTER	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && !CHECK_IN_MASTER_LIST => ADD_TO_MASTER_LIST StartupTimer.restart (MasterStartupTime)	T_MASTER
49	T_MASTER	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST && !VALID_ANNOUNCE => ignore	T_MASTER
50	T_MASTER	Announce_ind (PTCP_ANNOUNCE_PDU) /VALID_SYNC_MASTER && CHECK_IN_MASTER_LIST && VALID_ANNOUNCE => UPDATE_MASTER_ENTRY StartupTimer.restart (MasterStartupTime)	T_MASTER
51	T_MASTER	AnnTimer.expired () /!MASTER_LIST_EMPTY => SequenceID++ AnnTimer.start (AnnInterval) StartupTimer.restart (MasterStartupTime) Announce_req (PTCP_ANNOUNCE_PDU)	T_MASTER
52	T_MASTER	AnnTimer.expired () /MASTER_LIST_EMPTY => SequenceID++ AnnTimer.start (AnnInterval) Announce_req (PTCP_ANNOUNCE_PDU)	T_MASTER
53	T_MASTER	StartupTimer.expired () =>	MASTER

#	Current State	Event /Condition =>Action	Next State
54	T_MASTER	MasterListAgingT.expired () => AGING_MASTER_LIST MasterListAgingT.start (MASTER_AGING_TIME)	T_MASTER

Table 104 – BMA best remote sync master (RSM) state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	GetBestRSM (SyncID, MasterList) /MasterList.Entry{SyncID}.Num_entry == 0 => Best_RSM_SA := NIL	IDLE
2	IDLE	GetBestRSM (SyncID, MasterList) /MasterList.Entry{SyncID}.Num_entry == 1 => N_BestM := 0 Best_RSM_SA := MasterList.Entry{SyncID, N_BestM}.SA Result := OK	IDLE
3	IDLE	GetBestRSM (SyncID, MasterList) /MasterList.Entry{SyncID}.Num_entry > 1 => N_BestM := 0 N_CmpM := 1	CMP_R_P1
4	CMP_R_P1	/N_CmpM < MasterList.Entry{SyncID}.Num_entry && MasterList.Entry{SyncID}[N_BestM].Priority1 > MasterList.Entry{SyncID}[N_CmpM].Priority1 => N_CmpM++	CMP_R_P1
5	CMP_R_P1	/N_CmpM < MasterList.Entry{SyncID}.Num_entry && MasterList.Entry{SyncID}[N_BestM].Priority1 < MasterList.Entry{SyncID}[N_CmpM].Priority1 => N_BestM := N_CmpM N_CmpM++	CMP_R_P1
6	CMP_R_P1	/N_CmpM < MasterList.Entry{SyncID}.Num_entry && MasterList.Entry{SyncID}[N_BestM].Priority1 == MasterList.Entry{SyncID}[N_CmpM].Priority1 =>	CMP_R_A
7	CMP_R_P1	/N_CmpM >= MasterList.Entry{SyncID}.Num_entry => Best_RSM_SA := MasterList.Entry{SyncID, N_BestM }.SA	IDLE
8	CMP_R_A	/MasterList.Entry{SyncID}[N_BestM].Accuracy > MasterList.Entry{SyncID}[N_CmpM].Accuracy => N_CmpM++	CMP_R_P1
9	CMP_R_A	/MasterList.Entry{SyncID}[N_BestM].Accuracy < MasterList.Entry{SyncID}[N_CmpM].Accuracy => N_BestM := N_CmpM N_CmpM++	CMP_R_P1
10	CMP_R_A	/MasterList.Entry{SyncID}[N_BestM].Accuracy == MasterList.Entry{SyncID}[N_CmpM].Accuracy=>	CMP_R_V

#	Current State	Event /Condition =>Action	Next State
11	CMP_R_V	/N_CmpM < MasterList.Entry{SyncID}.Num_entry && MasterList.Entry{SyncID}[N_BestM].Variance > MasterList.Entry{SyncID}[N_CmpM].Variance => N_CmpM++	CMP_R_P1
12	CMP_R_V	/MasterList.Entry{SyncID}[N_BestM].Variance < MasterList.Entry{SyncID}[N_CmpM].Variance => N_BestM := N_CmpM N_CmpM++	CMP_R_P1
13	CMP_R_V	/MasterList.Entry{SyncID}[N_BestM].Variance == MasterList.Entry{SyncID}[N_CmpM].Variance =>	CMP_R_P2
14	CMP_R_P2	/N_CmpM < MasterList.Entry{SyncID}.Num_entry && MasterList.Entry{SyncID}[N_BestM].Priority2 > MasterList.Entry{SyncID}[N_CmpM].Priority2 => N_CmpM++	CMP_R_P1
15	CMP_R_P2	/N_CmpM < MasterList.Entry{SyncID}.Num_entry && MasterList.Entry{SyncID}[N_BestM].Priority2 < MasterList.Entry{SyncID}[N_CmpM].Priority2 => N_BestM := N_CmpM N_CmpM++	CMP_R_P1
16	CMP_R_P2	/N_CmpM < MasterList.Entry{SyncID}.Num_entry && MasterList.Entry{SyncID}[N_BestM].Priority2 == MasterList.Entry{SyncID}[N_CmpM].Priority2 =>	CMP_R_MA
17	CMP_R_P2	/N_CmpM >= MasterList.Entry{SyncID}.Num_entry <=> Best_RSM_SA := MasterList.Entry{SyncID, N_BestM }.SA	IDLE
18	CMP_R_MA	/MasterList.Entry{SyncID}[N_BestM].SA < MasterList.Entry{SyncID}[N_CmpM].SA => N_CmpM++	CMP_R_P1
19	CMP_R_MA	/MasterList.Entry{SyncID}[N_BestM].SA > MasterList.Entry{SyncID}[N_CmpM].SA=>N_BestM := N_CmpM N_CmpM++	CMP_R_P1
20	CMP_R_MA	/MasterList.Entry{SyncID}[N_BestM].SA == MasterList.Entry{SyncID}[N_CmpM].SA => N_CmpM++	CMP_R_P1

Table 105 – BMA get best sync master (GBSM) state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	GetBestSM (SM_1, SM_2) /SM_1 == NIL && SM_2.SA == NIL => BetterSM_SA := NIL	IDLE
2	IDLE	GetBestSM (SM_1, SM_2) /SM_1 == NIL && SM_2.SA != NIL => BetterSM_SA := SM_2.SA	IDLE
3	IDLE	GetBestSM (SM_1, SM_2) /SM_1 != NIL && SM_2.SA != NIL =>	CMP_R_L_P1
4	CMP_R_L_P1	/SM_1.Priority1 < SM_2.Priority1 => BetterSM_SA := SM_1.SA	IDLE
5	CMP_R_L_P1	/SM_1.Priority1 > SM_2.Priority1 => BetterSM_SA := SM_2.SA	IDLE
6	CMP_R_L_P1	/SM_1.Priority1 == SM_2.Priority1 =>	CMP_R_L_A
7	CMP_R_L_A	/SM_1.Accuracy < SM_2.Accuracy => BetterSM_SA := SM_1.SA	IDLE
8	CMP_R_L_A	/SM_1.Accuracy > SM_2.Accuracy => BetterSM_SA := SM_2.SA	IDLE
9	CMP_R_L_A	/SM_1.Accuracy == SM_2.Accuracy =>	CMP_R_L_V
10	CMP_R_L_V	/SM_1.Variance < SM_2.Variance => BetterSM_SA := SM_1.SA	IDLE
11	CMP_R_L_V	/SM_1.Variance > SM_2.Variance=>BetterSM_SA := SM_2.SA	IDLE
12	CMP_R_L_V	/SM_1.Variance == SM_2.Variance =>	CMP_R_L_P2
13	CMP_R_L_P2	/SM_1.Priority1 < SM_2.Priority1 => BetterSM_SA := SM_1.SA	IDLE
14	CMP_R_L_P2	/SM_1.Priority1 > SM_2.Priority1 =>	IDLE

#	Current State	Event /Condition =>Action	Next State
		BetterSM_SA := SM_2.SA	
15	CMP_R_L_P2	/SM_1.Priority1 == SM_2.Priority1 =>	CMP_R_L_MA
16	CMP_R_L_MA	/SM_1.SA < SM_2.SA => BetterSM_SA := SM_1.SA	IDLE
17	CMP_R_L_MA	/SM_1.SA > SM_2.SA => BetterSM_SA := SM_2.SA	IDLE
18	CMP_R_L_MA	/SM_1.SA == SM_2.SA => BetterSM_SA := SM_1.SA	IDLE

4.4.4.4.4 Macros

Table 106 contains the macros used by BMA.

Table 106 – Macros used by BMA

Name	Meaning
VALID_SYNC_MASTER	Domain is supported PTCP_ANNOUNCE_PDU.DomainUUID == DomainUUID
ini MasterList	initialize data set for remote master MasterList.Num_entry := 0
CHECK_IN_MASTER_LIST	Master in master list sm_sa := PTCP_ANNOUNCE_PDU.MasterSourceAddress sm_sa in MasterList
VALID_ANNOUNCE	Check announce frame sm_sa := PTCP_ANNOUNCE_PDU.MasterSourceAddress MasterList.Entry{sm_sa}.SequenceID <= PTCP_ANNOUNCE_PDU.SequenceID
ADD_TO_MASTER_LIST	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++
UPDATE_MASTER_ENTRY	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
REM_ENTRY	Remove master from remote secondary master list MasterList.Num_entry-- MasterList.Remove(sm_sa)
AGING_MASTER_LIST	Aging of entries in remote sync master list for m := 0 to MasterList.Num_entry if (MasterList.Entry[m].Receipt == 0

4.4.4.4.5 Functions

Table 107 contains the functions used by BMA.

Table 107 – Functions used by BMA

Name	Meaning
BMA_Start_req ()	Activate best master function
BMA_Start_cnf ()	Confirmation for best master function start
BMA_Stop_req ()	Deactivates best master function
BMA_Stop_cnf ()	Confirmation for stop best master function
BMA_State_ind (Role)	During the startup period the BMA elects the best sync master. Allowed value for Role: - MASTER - SLAVE
SLAVE_State_ind(Status)	Indication for master lost. The event is generated by the SLAVE. - MASTER_LOST
Announce_req ()	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 (Status)	LMPM_A_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status
Announce_ind (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

4.4.4.5 Master Protocol Machine (MPSM)

4.4.4.5.1 Primitive definitions

4.4.4.5.1.1 Primitives exchanged between MPSM and ASE

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

4.4.4.5.2 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 existing PTCP SyncID. Therefore an instance of the state machine exists for every SyncID. To achieve these transitions the MPSM has to initiate synchronization and announce messages. Announce messages shall be sent to advertise master capabilities and grant that only one master is active. Synchronization messages shall only send during the start up period.

After start up the data management shall be initialized (POWER-ON) and the synchronization role has to be set (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.

Multiple active primary masters shall be prevented by the state STARTUP. With the role master the MPSM sends several announce messages. Therefore the decision to go from state STARTUP to state MASTER (primary manager) will be made with extensive knowledge of concurrent masters.

If an active master (with fewer capabilities) already exists and after the start up period a new master comes along, the new master needs to synchronize to the active master. For synchronization the MPSM goes to state SLAVE.

Figure 33 shows the MPSM state machine:

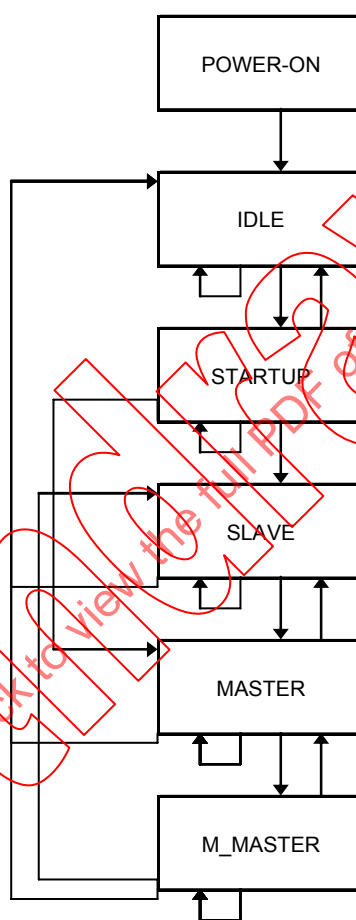


Figure 33 – State transition diagram of MPSM

States of the MPSM

POWER_ON

Data initialization

IDLE

The ASE service “StartMaster_req” shall start the master function of the device.

STARTUP

During the start up period the MPSM is in state STARTUP.

SLAVE

The master is synchronized by sync messages from an other 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 issue.

4.4.4.5.3 MPSM state table

Table 108 contains the state table used by MPSM.

Table 108 – 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_conf (Status)	STARTUP
3	IDLE	StopMaster_req () => Status := OK StopMaster_conf (Status)	IDLE
4	IDLE	Sync_cnf () (D_Port) =>	IDLE
5	IDLE	BMA_Stop_cnf () =>	IDLE
6	STARTUP	StartMaster_req () => Status := WRONG_SEQUENCE StartMaster_conf (Status)	STARTUP
7	STARTUP	StopMaster_req () => Status := OK StopMaster_conf (Status) BMA_Stop_req ()	IDLE
8	STARTUP	Sync_cnf () (D_Port) =>	STARTUP
9	STARTUP	BMA_State_ind (State) /State == SLAVE => Status := OK StartSlave_req ()	SLAVE
10	STARTUP	BMA_State_ind (State) /State == MASTER => State := SINGLE_MASTER SequenceID++ SyncTimer.start (SyncInterval) TimeoutT.start (SyncTimeOut) for i := all op. D_Ports do if !(EGRESS_BORDERLINE(D_Port)) Sync_req (S_Port, PTCP_SYNC_PDU)	MASTER
11	SLAVE	StartMaster_req () => Status := OK StartMaster_conf (Status) BMA_Start_req ()	SLAVE

#	Current State	Event /Condition =>Action	Next State
12	SLAVE	StopMaster_req () => Status := OK BMA_Stop_req () StopMaster_conf (Status)	IDLE
13	SLAVE	Sync_cnf () (D_Port)=>	SLAVE
14	SLAVE	BMA_State_ind (State) /State == SLAVE =>	SLAVE
15	SLAVE	BMA_State_ind (State) /State == MASTER => State := SINGLE_MASTER SequenceID++ SyncTimer.start (SyncInterval) TimeoutT.start (SyncTimeOut) StopSlave_req () for i := all op. D_Ports do if !(EGRESS_BORDERLINE(D_Port)) Sync_req (i, PTCP_SYNC_PDU)	MASTER
16	MASTER	StartMaster_req () => Status := WRONG_SEQUENCE StartMaster_conf (Status)	MASTER
17	MASTER	StopMaster_req () => Status := OK BMA_Stop_req () StopMaster_conf (Status)	IDLE
18	MASTER	BMA_State_ind (State) /State == SLAVE => SyncTimer.stop () TimeoutT.stop () StartSlave_req ()	SLAVE
19	MASTER	BMA_State_ind (State) /State == MASTER =>	MASTER
20	MASTER	Sync_ind (-) (S_Port, PTCP_SYNC_PDU) => ignore	MASTER
21	MASTER	Sync_ind (+) (S_Port, PTCP_SYNC_PDU) /VALID_SYNC =>	MASTER
22	MASTER	Sync_ind (+) (S_Port, PTCP_SYNC_PDU) /VALID_SYNC && INGRESS_BORDERLINE =>	MASTER
23	MASTER	Sync_ind (+) (S_Port, PTCP_SYNC_PDU) /VALID_SYNC && !INGRESS_BORDERLINE => MultipleMaster := TRUE	MASTER
24	MASTER	Sync_cnf (-) (D_Port) => Sync_req (D_Port, PTCP_SYNC_PDU)	MASTER
25	MASTER	Sync_cnf (+) (D_Port)=>	MASTER

#	Current State	Event /Condition =>Action	Next State
26	MASTER	SyncTimer.expired () => SequenceID++ SyncTimer.start (SyncInterval) for i := all op. D_Ports do if !(EGRESS_BORDERLINE(D_Port)) Sync_req (i, PTCP_SYNC_PDU)	MASTER
27	MASTER	TimeoutT.expired () /MultipleMaster => TimeoutT.start (SyncTimeOut)	MASTER
28	MASTER	TimeoutT.expired () /MultipleMaster => MultipleMaster := FALSE TimeoutT.start (SyncTimeOut) SyncStateChange_ind (MULTIPLE_MASTER)	M_MASTER
29	M_MASTER	StartMaster_req () => Status := WRONG_SEQUENCE StartMaster_conf (Status)	M_MASTER
30	M_MASTER	StopMaster_req () => Status := OK BMA_Stop_req () StopMaster_conf (Status)	IDLE
31	M_MASTER	BMA_State_ind (State) /State == SLAVE => SyncTimer.stop () TimeoutT.stop () StartSlave_req ()	SLAVE
32	M_MASTER	BMA_State_ind (State) /State == MASTER =>	M_MASTER
33	M_MASTER	Sync_ind (-) (S_Port, PTCP_SYNC_PDU) => ignore	M_MASTER
34	M_MASTER	Sync_ind (+) (S_Port, PTCP_SYNC_PDU) /INVALID_SYNC =>	M_MASTER
35	M_MASTER	Sync_ind (+) (S_Port, PTCP_SYNC_PDU) /INVALID_SYNC && INGRESS_BORDERLINE =>	M_MASTER
36	M_MASTER	Sync_ind (+) (S_Port, PTCP_SYNC_PDU) /INVALID_SYNC && !INGRESS_BORDERLINE => MultipleMaster := TRUE	M_MASTER
37	M_MASTER	Sync_cnf (-) (D_Port) => Sync_req (D_Port, PTCP_SYNC_PDU)	M_MASTER
38	M_MASTER	Sync_cnf (+) (D_Port) =>	M_MASTER
39	M_MASTER	SyncTimer.expired () => SequenceID++ SyncTimer.start (SyncInterval) for i := all op. D_Ports do if !(EGRESS_BORDERLINE(D_Port)) Sync_req (i, PTCP_SYNC_PDU)	M_MASTER

#	Current State	Event /Condition =>Action	Next State
40	M_MASTER	TimeoutT.expired () /!MultipleMaster => TimeoutT.start (SyncTimeOut) SyncStateChange_ind (SINGLE_MASTER)	MASTER
41	M_MASTER	TimeoutT.expired () /MultipleMaster => MultipleMaster := FALSE TimeoutT.start (SyncTimeOut)	M_MASTER

4.4.4.5.4 Macros

Table 109 contains the macros used by MPSM.

Table 109 – Macros used by MPSM

Name	Meaning
VALID_SYNC	Check for received sync frame PTCP_SYNC_PDU.DomainUUID == DomainUUID PTCP_SYNC_PDU.MasterSourceAddress != local master address PTCP_SYNC_PDU.SequenceID > stored remote master sequenceID
EGRESS_BOUNDARY	Egress borderline for sync frames
INGRESS_BORDERLINE	Ingress borderline for sync frames

4.4.4.5.5 Functions

Table 110 contains the functions used by MPSM.

Table 110 – Functions used by MPSM

Name	Meaning
StartMaster_req ()	Activate master function
StartMaster_cnf ()	Confirmation for master start
StopMaster_req ()	Deactivates master function
StopMaster_cnf ()	Confirmation for master stop
StartSlave_req ()	Activate slave function
StartSlave_cnf ()	Confirmation for start slave
StopSlave_req ()	Deactivates slave function
StopSlave_cnf ()	Confirmation for stop slave
BMA_Start_req ()	Activate best master function
BMA_Start_cnf ()	Confirmation for best master function start
BMA_Stop_req ()	Deactivates best master function
BMA_Stop_cnf ()	Confirmation for stop best master function
BMA_State_ind (Role)	During the startup period the BMA elects the best sync master. Allowed value for Role: - MASTER - SLAVE
SyncStateChange_ind(Status)	Indication for events - SINGLE_MASTER - MULTIPLE_MASTER
Sync_req (D_Port, PTCP_SYNC_PDU)	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)
Sync_cnf (Status, D_Port)	LMPM_P_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status
Sync_ind (PTCP_SYNC_PDU)	Receive PTCP-PDU according PTCP_SYNC_PDU LMPM_P_Data.ind (CREP, S_Port, TSamp, DA, SA, A_SDU) Assignments: PTCP_SYNC_PDU := A_SDU without LT and FrameID

4.4.4.6 Slave Protocol Machine (SPSM)

4.4.4.6.1 Primitive definitions

4.4.4.6.1.1 Primitives exchanged between SPSM and ASE

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

4.4.4.6.2 State machine description

The slave protocol machine controls the transition of a slave device. Therefore an instance of the state machine exists for every SyncID.

After start up the data management shall be initialized (POWER-ON) and the synchronization role has to 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.

Figure 34 shows the SPSM state machine:

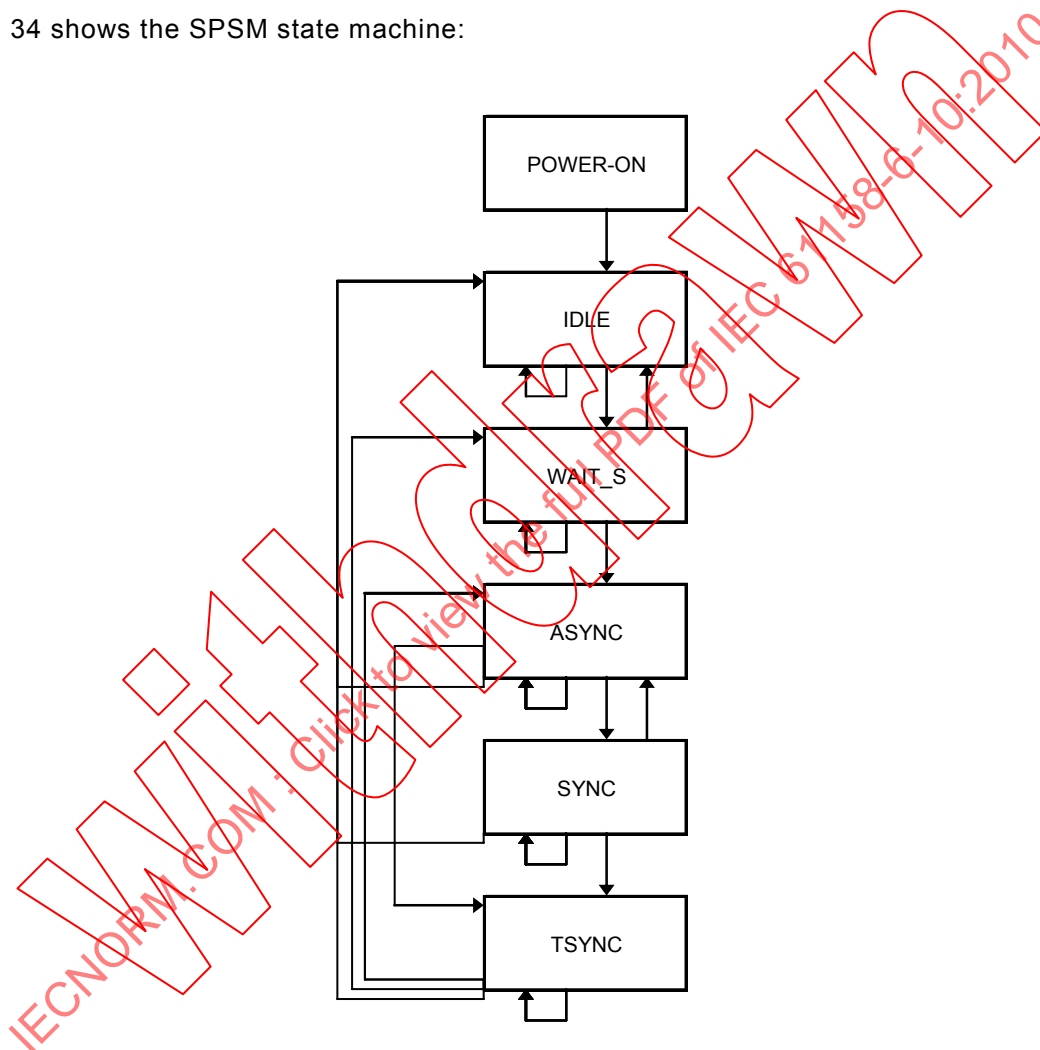


Figure 34 – State transition diagram of SPSM

States of the SPSM

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.

ASYN

The slave is not synchronized to the active master of the sync domain..

SYN

The slave has achieved synchronity with the activ master of the sync domain.

TSYN

The slave is still waiting for new sync messages.

4.4.4.6.3 SPSM state table

Table 111 contains the state table used by SPSM.

Table 111 – SPSM state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> M_SA := NIL	IDLE
2	IDLE	StartSlave_req () => store Parameter Status := OK TakeoverTimeoutT.start (MasterTimeOut) StartSlave_cnf (Status)	WAIT_S
3	IDLE	StopSlave_req () => Staus := OK StopSlave_cnf (Status)	IDLE
4	WAIT_S	StartSlave_req () => Staus := WRONG_SEQUENCE StartSlave_cnf (Status)	WAIT_S
5	WAIT_S	StopSlave_req () => TakeoverTimeoutT.stop () Staus := OK StopSlave_cnf (Status)	IDLE
6	WAIT_S	Sync_ind (-) (S_Port, LocalTime, PTCP_SYNC_DATA) => ignore	WAIT_S
7	WAIT_S	Sync_ind (-) (S_Port, LocalTime, PTCP_SYNC_DATA) /INVALID_DOMAIN_ID =>	WAIT_S
8	WAIT_S	Sync_ind (-) (S_Port, LocalTime, PTCP_SYNC_DATA) /INVALID_DOMAIN_ID && INGRESS_BORDERLINE =>	WAIT_S
9	WAIT_S	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /INVALID_DOMAIN_ID && !INGRESS_BORDERLINE => L_Time := LocalTime M_Time := PTCP_SYNC_DATA.Time M_Seq := PTCP_SYNC_DATA.SequencelD M_SA := PTCP_SYNC_DATA.MasterSourceAddress OFFSET_CTRL_START (L_Time, M_Time) OffsetContLoopT.start (SyncInterval) TakeoverTimeoutT.restart (MasterTimeOut) SyncTimeoutT.start (SyncTimeOut) SyncStateChange_ind (JITTER_OUT_OF_BOUNDARY)	ASYN

#	Current State	Event /Condition =>Action	Next State
10	WAIT_S	TakeoverTimeoutT.expired () => M_SA := NIL SLAVE_State_ind (MASTER_LOST)	WAIT_S
11	ASYNC	StartSlave_req () => Staus := WRONG_SEQUENCE StartSlave_cnf (Status)	ASYNC
12	ASYNC	StopSlave_req () => OFFSET_CTRL_STOPOffsetContLoopT.stop ()TakeoverTimeoutT.stop ()SyncTimeoutT.stop ()Staus := OKStopSlave_cnf (Status)	IDLE
13	ASYNC	Sync_ind (-) (S_Port, LocalTime, PTCP_SYNC_DATA) => ignore	ASYNC
14	ASYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /!VALID_DOMAIN_ID => ignore	ASYNC
15	ASYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /VALID_DOMAIN_ID && INGRESS_BORDERLINE => ignore	ASYNC
16	ASYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /VALID_DOMAIN_ID && !INGRESS_BORDERLINE && !VALID_SYNC_MASTER => ignore	ASYNC
17	ASYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /VALID_DOMAIN_ID && !INGRESS_BORDERLINE && VALID_SYNC_MASTER => L_Time := LocalTime M_Time := PTCP_SYNC_DATA.Time M_Seq := PTCP_SYNC_DATA.SequencelD TakeoverTimeoutT.restart (MasterTimeOut) SyncTimeoutT.restart (SyncTimeOut)	ASYNC
18	ASYNC	OffsetContLoopT.expired () /OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (L_Time, M_Time) OffsetContLoopT.start (SyncInterval)	ASYNC
19	ASYNC	OffsetContLoopT.expired () /OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (L_Time, M_Time) OffsetContLoopT.start (SyncInterval) SyncStateChange_ind (SYNC)	SYNC
20	ASYNC	TakeoverTimeoutT.expired () => M_SA := NIL SLAVE_State_ind (MASTER_LOST)	TSYNC
21	SYNC	StartSlave_req () => Staus := WRONG_SEQUENCE StartSlave_cnf (Status)	SYNC

#	Current State	Event /Condition =>Action	Next State
22	SYNC	StopSlave_req () => OFFSET_CTRL_STOP OffsetContLoopT.stop () TakeoverTimeoutT.stop () SyncTimeoutT.stop () Staus := OK StopSlave_cnf (Status)	IDLE
23	SYNC	Sync_ind (-) (S_Port, LocalTime, PTCP_SYNC_DATA) => ignore	SYNC
24	SYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /INVALID_DOMAIN_ID =>	SYNC
25	SYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /INVALID_DOMAIN_ID && INGRESS_BORDERLINE => ignore	SYNC
26	SYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /INVALID_DOMAIN_ID && !INGRESS_BORDERLINE && !INVALID_SYNC_MASTER => ignore	SYNC
27	SYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /INVALID_DOMAIN_ID && !INGRESS_BORDERLINE && VALID_SYNC_MASTER => L_Time := LocalTime M_Time := PTCP_SYNC_DATA.Time M_Seq := PTCP_SYNC_DATA.SequenceID TakeoverTimeoutT.restart (MasterTimeOut) SyncTimeoutT.restart (SyncTimeOut)	SYNC
28	SYNC	OffsetContLoopT.expired () /OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (L_Time, M_Time) OffsetContLoopT.start (SyncInterval)	SYNC
29	SYNC	OffsetContLoopT.expired () /!OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (L_Time, M_Time) OffsetContLoopT.start (SyncInterval) SyncStateChange_ind (JITTER_OUT_OF_BOUNDARY)	ASYN
30	SYNC	TakeoverTimeoutT.expired () => M_SA := NIL SLAVE_State_ind (MASTER_LOST)	TSYN
31	TSYN	StartSlave_req () => Staus := WRONG_SEQUENCE StartSlave_cnf (Status)	TSYN
32	TSYN	StopSlave_req () => OFFSET_CTRL_STOP OffsetContLoopT.stop () TakeoverTimeoutT.stop () SyncTimeoutT.stop () Staus := OK StopSlave_cnf (Status)	IDLE

#	Current State	Event /Condition =>Action	Next State
33	TSYNC	Sync_ind (-) (S_Port, LocalTime, PTCP_SYNC_DATA) => ignore	TSYNC
34	TSYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /!VALID_DOMAIN_ID => ignore	TSYNC
35	TSYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /VALID_DOMAIN_ID && INGRESS_BORDERLINE => ignore	TSYNC
36	TSYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /VALID_DOMAIN_ID && !INGRESS_BORDERLINE && !VALID_SYNC_MASTER => ignore	TSYNC
37	TSYNC	Sync_ind (+) (S_Port, LocalTime, PTCP_SYNC_DATA) /VALID_DOMAIN_ID && !INGRESS_BORDERLINE && VALID_SYNC_MASTER => L_Time := LocalTime M_Time := PTCP_SYNC_DATA.Time M_Seq := PTCP_SYNC_DATA.SequenceID TakeoverTimeoutT.restart (MasterTimeOut) SyncTimeoutT.restart (SyncTimeOut)	ASync
38	TSYNC	OffsetContLoopT.expired () => OffsetContLoopT.start (SyncInterval)	TSYNC
39	TSYNC	SyncTimeoutT.expired () => OffsetContLoopT.stop () OFFSET_CTRL_STOP SyncStateChange_ind (NO_SYNC_MESSAGE_RECEIVED)	WAIT_S

4.4.4.6.4 Macros

Table 112 contains the macros used by SPSM.

Table 112 – Macros used by SPSM

Name	Meaning
VALID_DOMAIN_ID	Check domain UUID from received sync frame PTCP_SYNC_PDU.DomainUUID == DomainUUID
VALID_SYNC_MASTER	Received sync frame from present master PTCP_SYNC_PDU.SequenceID > M_Seq PTCP_SYNC_PDU.MasterSourceAddress == M_SA
OFFSET_CTRL_START (L_Time, M_Time)	Initialize control loop for offset compensation
OFFSET_CTRL_CALC (L_Time, M_Time)	Offset compensation
OFFSET_CTRL_STOP	Stop offset compensation
OFFSET_CTRL_IS_SYNC	Measured sync error in PLL Window
INGRESS_BORDERLINE	Ingress borderline for sync frames
VALID_DOMAIN_ID	Check domain UUID from received sync frame PTCP_SYNC_PDU.DomainUUID == DomainUUID
VALID_SYNC_MASTER	Received sync frame from present master

4.4.4.6.5 Functions

Table 113 contains the functions used by SPSM.

Table 113 – Functions used by SPSM

Name	Meaning
StartSlave_req ()	Activate slave function
StartSlave_cnf ()	Confirmation for start slave
StopSlave_req ()	Deactivates slave function
StopSlave_cnf ()	Confirmation for stop slave
SyncStateChange_ind(Status)	Indication for events - JITTER_OUT_OF_BOUNDARY - NO_SYNC_MESSAGE_RECEIVED - WRONG_PTCP_DOMAIN
SLAVE_State_ind(Status)	Indication for events used by the best master algorithmus - MASTER_LOST
Sync_ind (S_Port, TStamp, PTCP_SYNC_PDU)	Receive PTCP-PDU according PTCP_SYNC_PDU LMPM_P_Data.ind (CREP, S_Port, TSamp, DA, SA, A_SDU) Assignments: PTCP_SYNC_PDU := A_SDU without LT and FrameID

4.4.4.7 Sync Relay Protocol Machine (SRPM)

4.4.4.7.1 Primitive definitions

4.4.4.7.1.1 Primitives exchanged between SRPM and ASE

4.4.4.7.1.2 Parameters of SRPM primitives

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

4.4.4.7.2 State machine description

The sync relay protocol machine describes sync forwarding rules for bridges. The description of the sync relay protocol machine assumes that the time stamp unit of the brige can corret the bridging delay of sync frames on the fly. The time correction (bridge delay and line delay) is done be transmitting a sync frame. The sync relay protocol machine is necessary to avoid circulating sync messages in the network.

Figure 35 shows the SRPM state machine:

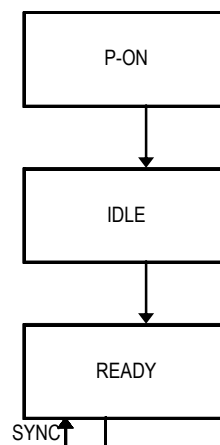


Figure 35 – State transition diagram of SRPM

States of the SRPM

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 followup messages.

4.4.4.7.3 SRPM state table

Table 114 contains the state table used by SRPM. 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. The Table 117 shows at a glance the conditions.

Table 114 – SRPM state table

#	Current State	Event /Condition =>Action	Next State
1	P-ON	=> Ini RelayTable	IDLE
2	IDLE	=> ReceiptTimer.start (PTCP_Timeout, PTCP_TakeoverTimeout)	READY
3	READY	ReceiptTimer.expired (PTCP_Timeout) => AGING_OF_FWSM_LIST(PT) ReceiptTimer.start (PTCP_Timeout)	READY
4	READY	ReceiptTimer.expired (PTCP_TakeoverTimeout) => AGING_OF_FWSM_LIST(PTT) ReceiptTimer.start (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 && !RATE_VALID && !NEW_SYNC_FRAME => ignore	READY
7	READY	Sync_Ind (+) (S_Port, PTCP_SYNC_PDU) /RECEIVE_PORT_VALID && !RATE_VALID && NEW_SYNC_FRAME => UPDATE_MASTER_ENTRY CALC_MASTER_RCF	READY
8	READY	Sync_Ind (+) (S_Port, PTCP_SYNC_PDU) /RECEIVE_PORT_VALID && RATE_VALID && NEW_SYNC_FRAME => UPDATE_MASTER_ENTRY CALC_MASTER_RCF for i = all op. D_Ports \ S_Port do if (TRANSMIT_PORT_VALID) Sync_Req (i, PTCP_SYNC_PDU)	READY
9	READY	Sync_Cnf () () => ignore	READY
10	READY	Followup_Ind (+) (S_Port, PTCP_FUP_PDU) /RECEIVE_PORT_VALID => ignore	READY
11	READY	Followup_Ind (+) (S_Port, PTCP_FUP_PDU) /RECEIVE_PORT_VALID&& !VALID_FU_FRAME => ignore	READY

#	Current State	Event /Condition =>Action	Next State
12	READY	Followup_Ind (+) (S_Port, PTCP_FUP_PDU) /RECEIVE_PORT_VALID && RATE_VALID && VALID_FU_FRAME => 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

4.4.4.7.4 Macros

Table 115 contains the macros used by SRPM.

Table 115 – Macros used by SRPM

Name	Function
RECEIVE_PORT_VALID	The PortState, the MAUType, the LineDelay, the IngressBoundary, the PTCP_SubdomainUUID and the MasterMACAddress shall allow the reception of the PTCP frames.
TRANSMIT_PORT_VALID	The PortState, the MAUType, the LineDelay, the EgressBoundary, the PTCP_SubdomainUUID compared with the neighborhood information and the MasterMACAddress compared with the neighborhood information shall allow the transmission of the PTCP frames.
UPDATE_MASTER_ENTRY	Update entry in RelayTable for sync master sm_sa := PTCP_SYNC_PDU.MasterSourceAddress SyncID:= SYNC_ID_MASK (FrameID) RelayTable.Entry{sm_sa, SyncID}.seq := PTCP_SYNC_PDU.SequenceID RelayTable.Entry{sm_sa, SyncID}.RcvPort := S_Port RelayTable.Entry{sm_sa, SyncID}.FollowUp := FALSE RelayTable.Entry{sm_sa, SyncID}.Receipt++
AGING_OF_FWSM_LIST()	Aging of Entries in RelayTable PTT: Delete the stored MasterMACAddress to enable the receive of the sync frame from a primary/secondary master PT: Delete the stored MasterMACAddress, the stored PTCP_SubdomainUUID (if not defined locally) and set the rate invalid
NEW_SYNC_FRAME	Check receive port and sequence number is valid
CALC_MASTER_RCF	Update the calculated rate
VALID_FU_FRAME	Check receive port and sequence number is valid sm_sa := PTCP_FUP_PDU.MasterSourceAddress SyncID:= SYNC_ID_MASK (FrameID) RelayTable.Entry{ sm_sa, SyncID}.RcvPort == S_Port RelayTable.Entry{ sm_sa, SyncID}.seq == PTCP_FUP_PDU.SequenceID RelayTable.Entry{sm_sa, SyncID}.FollowUp == FALSE
SET_FU_RECEIVED	Update entry in RelayTable sm_sa := PTCP_FUP_PDU.MasterSourceAddress SyncID:= SYNC_ID_MASK (FrameID) RelayTable.Entry{sm_sa, SyncID}.FollowUp := TRUE

4.4.4.7.5 Functions

Table 116 contains the functions used by the SRPM.

Table 116 – Functions used by the SRPM

Name	Function
Sync_Req (D_Port, PTCP_SYNC_PDU)	Create PTCP-PDU according PTCP_SYNC_PDU Assignments: D_Port := Port DA := multicast address fo 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)
Sync_Cnf (D_Port, Status)	LMPM_P_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status
Sync_Ind (S_Port, PTCP_SYNC_PDU)	Receive PTCP-PDU according PTCP_SYNC_PDU LMPM_P_Data.ind (CREP, D_Port, TSamp, DA, SA, A_SDU) Assignments: PTCP_SYNC_PDU := A_SDU without LT and FrameID
Followup_Req (D_Port, PTCP_FUP_PDU)	Create PTCP-PDU according PTCP_FUP_PDU Assignments: D_Port := Port DA := multicast address fo 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)
Followup_Cnf (D_Port, Status)	LMPM_P_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status
Followup_Ind (S_Port, PTCP_FUP_PDU)	Receive PTCP-PDU according PTCP_FUP_PDU LMPM_P_Data.ind (CREP, D_Port, TSamp, DA, SA, A_SDU) Assignments: PTCP_FUP_PDU := A_SDU without LT and FrameID

4.4.4.7.6 Sync forwarding rules for bridges truth table for one SyncID

Table 117 contains the truth table used by the SRPM.

Table 117 – Truth table for one SyncID

Conditions of the receive port				Condition of the receive and the transmit port ^b		Local	Conditions of a transmit port				
PortState	MAU-Type	Line-Delay	Ingress-Filter	Subdomain-UUID	Master-MAC-Address	Rate known	PortState	MAU-Type	Line-Delay	Egress-Filter	Forwarding of a sync frame
Discard-ing	—	—	—	—	—	—	—	—	—	—	No
!Discard-ing	Wrong	—	—	—	—	—	—	—	—	—	No
!Discard-ing	Ok	No	—	—	—	—	—	—	—	—	No
!Discard-ing	Ok	Yes	Yes	—	—	—	—	—	—	—	No
!Discard-ing	Ok	Yes	No	Wrong	—	—	—	—	—	—	No
!Discard-ing	Ok	Yes	No	Equal ^a	Wrong	—	—	—	—	—	No
!Discard-ing	Ok	Yes	No	Equal ^a	Equal ^a	No	—	—	—	—	No
!Discard-ing	Ok	Yes	No	Equal ^a	Equal ^a	Yes	Discard-ing	—	—	—	No
!Discard-ing	Ok	Yes	No	Equal ^a	Equal ^a	Yes	!Discard-ing	Wrong	—	—	No
!Discard-ing	Ok	Yes	No	Equal ^a	Equal ^a	Yes	!Discard-ing	Ok	No	—	No
!Discard-ing	Ok	Yes	No	Equal ^a	Equal ^a	Yes	!Discard-ing	Ok	Yes	Yes	No
!Discard-ing	Ok	Yes	No	Equal ^a	Equal ^a	Yes	!Discard-ing	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 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. 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 SCHEDULER and ASE

4.4.4.8.1.2 Parameters of SCHEDULER primitives

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

4.4.4.8.2 State machine description

The SCHEDULER generates the signals for the local handling of time events.

Figure 36 shows the SCHEDULER state machine:

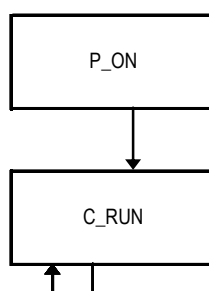


Figure 36 – State transition diagram of SCHEDULER

States of the SCHEDULER

P_ON

Data initialization

C_RUN

PTCP-Bridge with active rate-control

4.4.4.8.3 SCHEDULER state table

Table 118 contains the state table used by SCHEDULER.

Table 118 – SCHEDULER state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	=> Tx_Period := GREEN Rx_Period := GREEN YELLOW_TransPeriod := CycleLength - MAX_PACK_TANSMIT_TIME MUX_Schedule_req(Tx_Period) DMUX_Schedule_req(Rx_Period)	C_RUN
2	C_RUN	TickEvent (CycleCounter, CycleOffset) /Tx_Period <> RED && CycleOffset < Tx_RED_TransPeriod && Tx_RTC3_PortState == RUN => Tx_Period := RED MUX_Schedule_req(CycleCounter, Tx_Period)	C_RUN
3	C_RUN	TickEvent (CycleCounter, CycleOffset) /Tx_Period <> RED && CycleOffset < Tx_RED_TransPeriod && Tx_RTC3_PortState <> RUN => Tx_Period := RED	C_RUN
4	C_RUN	TickEvent (CycleCounter, CycleOffset) /Tx_Period <> ORANGE && CycleOffset >= Tx_RED_TransPeriod && Tx_RTC2_PortState == RUN => Tx_Period := ORANGE MUX_Schedule_req(CycleCounter, Tx_Period)	C_RUN
5	C_RUN	TickEvent (CycleCounter, CycleOffset) /Tx_Period <> ORANGE && CycleOffset >= Tx_RED_TransPeriod && Tx_RTC2_PortState <> RUN => Tx_Period := ORANGE	C_RUN
6	C_RUN	TickEvent (CycleCounter, CycleOffset) /Tx_Period <> GREEN && CycleOffset >= Tx_ORANGE_TransPeriod => Tx_Period := GREEN MUX_Schedule_req(CycleCounter, Tx_Period)	C_RUN
7	C_RUN	TickEvent (CycleCounter, CycleOffset) /Tx_Period <> YELLOW && CycleOffset >= YELLOW_TransPeriod => Tx_Period := YELLOW MUX_Schedule_req(CycleCounter, Tx_Period)	C_RUN
8	C_RUN	MUX_Schedule_cnf() => ignore	C_RUN
9	C_RUN	TickEvent (CycleCounter, CycleOffset) /Rx_Period <> RED && CycleOffset < Rx_RED_TransPeriod && Rx_RTC3_PortState == RUN => Rx_Period := RED DMUX_Schedule_req(CycleCounter, Rx_Period)	C_RUN
10	C_RUN	TickEvent (CycleCounter, CycleOffset) /Rx_Period <> RED&& CycleOffset < Rx_RED_TransPeriod&& Rx_RTC3_PortState <> RUN=>Rx_Period := RED	C_RUN
11	C_RUN	TickEvent (CycleCounter, CycleOffset) /Tx_Period <> GREEN && CycleOffset >= Rx_RED_TransPeriod => Rx_Period := GREEN DMUX_Schedule_req(CycleCounter, Rx_Period)	C_RUN

#	Current State	Event /Condition =>Action	Next State
12	C_RUN	DMUX_Schedule_cnf() => ignore	C_RUN
13	C_RUN	TickEvent (CycleCounter, CycleOffset) /for each active PPM check whether CycleCounter, CycleOffset fits to FrameSendOffset and ReductionRatio => C_Data_req()	C_RUN

4.4.4.8.4 Macros

SCHEDULER defines no macros.

4.4.4.8.5 Functions

Table 119 contains the functions used by the SCHEDULER, their arguments and their descriptions.

Table 119 – Functions used by the SCHEDULER

Function name	Description
C_Data_req()	if (txOption == RTC) LMPM_C_Data.req (CREP, PortList, DA, SA, Prio, C_SDU, APDU_Status) if (txOption == UDP) UDP_C_Data.req (IPPort, CREP, DA, SA, Prio, C_SDU, APDU_Status)

4.5 Media redundancy

4.5.1 FAL syntax description for MRP

4.5.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.5.1.2 APDU abstract syntax

Table 120 defines the abstract syntax of the MRP-PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs.

Table 120 – MRP APDU syntax

APDU name	APDU structure
MRP-PDU	MRP_Version (=0x0001), MRP_Type, MRP_Common, [MRP_Option], MRP_End, [Padding*] ^a
^a If the frame is shorter than 64 octets it will be extended with padding to 64 octets.	

Table 121 defines structures for substitutions of elements of the APDU structures shown in Table 120.

Table 121 – MRP substitutions

Substitution name	Structure
MRP_Type	MRP_Test ^ MRP_LinkChange ^ MRP_TopologyChange ^ MRP_Option
MRP_Common	MRP_TLVHeader, MRP_SequenceID, MRP_DomainUUID
MRP_Option	MRP_TLVHeader, MRP_ManufacturerOUI, Data*, [Padding*] ^a
MRP_End	MRP_TLVHeader (=0x0000)
MRP_Test	MRP_TLVHeader, MRP_Prio, MRP_SA ^b , MRP_PortRole, MRP_RingState, MRP_Transition, MRP_TimeStamp, [Padding*] ^a
MRP_TopologyChange	MRP_TLVHeader, MRP_Prio, MRP_SA ^b , MRP_Interval, [Padding*] ^a
MRP_LinkChange	MRP_LinkUp ^ MRP_LinkDown
MRP_LinkDown	MRP_TLVHeader, MRP_SA ^b , MRP_PortRole, MRP_Interval, MRP_Blocked, [Padding*] ^a
MRP_LinkUp	MRP_TLVHeader, MRP_SA ^b , MRP_PortRole, MRP_Interval, MRP_Blocked, [Padding*] ^a
^a 32bit alignment shall be ensured	
^b Shall be the interface MAC address	

4.5.1.3 Coding of the field MRP_TLVHeader

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: MRP_TLVHeader.Length

The value contains the sum of subsequent octets of the according block.

Bit 8 – 15: MRP_TLVHeader.Type

This field shall be coded with the values according to Table 122.

Table 122 – MRP_TLVHeader.Type

Value (hexadecimal)	Meaning	Usage
0x00	MRP_End (MRP_TLVHeader.Length shall be set to zero)	Mandatory
0x01	MRP_Common	Mandatory
0x02	MRP_Test	Mandatory
0x03	MRP_TopologyChange	Mandatory
0x04	MRP_LinkDown	Mandatory
0x05	MRP_LinkUp	Mandatory
0x06 – 0x7E	Reserved	—
0x7F	MRP_Option (Organizationally specific)	Optional

4.5.1.4 Coding of the field MRP_Version

This field shall be coded as data type Unsigned16. This field shall be set to one.

4.5.1.5 Coding of the field MRP_SequenceID

This field shall be coded as Unsigned16. It is used to identify the duplication of MRP frames in the ring. The range is from 0 to 65 535. The requesting application process shall provide an unique sequence number to each outstanding service request.

4.5.1.6 Coding of the field MRP_SA

This field shall be coded as data type OctetString[6]. The value of the field MRP_SA shall be a MAC address according to IEEE 802.

4.5.1.7 Coding of the field MRP_Prio

This field shall be coded as Unsigned16 and set according to Table 123.

Table 123 – MRP_Prio

Value (hexadecimal)	Meaning
0x0000	Highest priority redundancy manager
0x1000 – 0x7000	High priorities
0x8000	Default priority for redundancy manager
0x9000 – 0xE000	Low priorities
0xF000	Lowest priority redundancy manager
other	Reserved

4.5.1.8 Coding of the field MRP_PortRole

This field shall be coded as data type Unsigned16. The coding shall be according to Table 124.

Table 124 – MRP_PortRole

Value (hexadecimal)	Meaning	Usage
0x0000	Primary ring port	Frame is sent on primary ring port
0x0001	Secondary ring port	Frame is sent on backup ring port
0x0002 – 0xFFFF	Reserved	—

4.5.1.9 Coding of the field MRP_RingState

This field shall be coded as Unsigned16 with the values according to Table 125.

Table 125 – 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	—

4.5.1.10 Coding of the field MRP_Interval

This field shall be coded as Unsigned16 with the values according to Table 126.

Table 126 – MRP_Interval

Value (hexadecimal)	Meaning	Usage
0x0000 – 0x07D0	Interval for next topologie change event (in ms)	Mandatory
0x07D1 – 0xFFFF	Interval for next topologie change event (in ms)	Optional

4.5.1.11 Coding of the field MRP_Transition

This field shall be coded as Unsigned16 with the values according to Table 127.

Table 127 – MRP_Transition

Value (hexadecimal)	Meaning	Usage
0x0000 – 0xFFFF	Number of transitions from media redundancy ok to media redundancy lost	Used for monitoring this value via a packet sniffer station

4.5.1.12 Coding of the field MRP_TimeStamp

This field shall be coded as Unsigned32 with the values according to Table 128.

Table 128 – MRP_TimeStamp

Value (hexadecimal)	Meaning	Usage
0x00000000 – 0xFFFFFFFF	Actual counter value of 1 ms counter	The value is used by the MRM to determine the maximum travel time of the test frames in a ring.

4.5.1.13 Coding of the field MRP_Blocked

This field shall be coded as data type Unsigned16. This field shall be set to one.

4.5.1.14 Coding of the field MRP_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>>

4.5.1.15 Coding of the field MRP_DomainUUID

This field shall be coded as UUID with the values according to Table 129.

Table 129 – MRP_DomainUUID

Value (UUID)	Meaning	Usage
00000000-0000-0000-0000-000000000000	—	Reserved
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFFFFE	UUID for MRP redundancy domain	Optional
FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFFFFF	Default UUID for MRP redundancy domain	Mandatory

4.5.1.16 Coding of the field MRP_LengthDomainName

This field shall be coded as data type Unsigned8.

4.5.1.17 Coding of the field MRP_DomainName

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 MRP_DomainName is not terminated by zero.

4.5.2 FAL Service Protocol Machines (FSPMs)

4.5.2.1 MRM Protocol Machine for MRP

Figure 37 shows the principal behavior of the protocol machine for a media redundancy manager MRM.

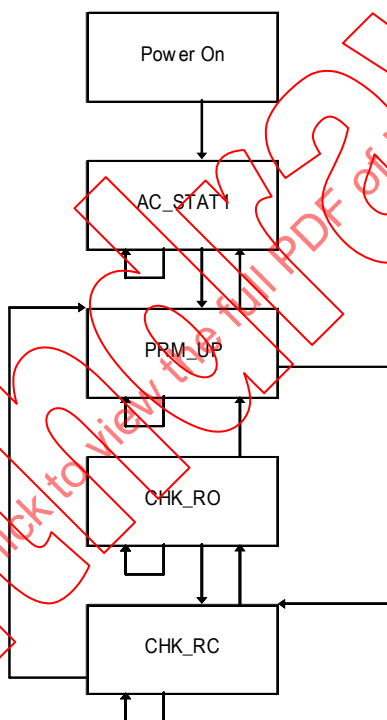


Figure 37 – MRM protocol machine for MRP

The protocol machine states perform in general the following tasks:

Power On

Initialization, the MRM shall start with both ring ports RPort_1 and RPort_2 in the port state BLOCKED. Static FDB entries for MRP multicast addresses MC_TEST and MC_CONTROL to host are generated. All MRP-PDU have highest priority (ORG).

AC_STAT1

Startup, waiting for the first Link Up at one of its ring ports (primary ring port), starting test monitoring of the ring and transition to PRM_UP.

PRM_UP (Primary Ring Port with Link Up)

This state shall be reached if only the primary ring port has a link (secondary ring port with no link). The MRM shall send test frames periodically thru both ring ports.

CHK_RO (Check Ring, Ring Open State)

The MRM didn't receive its test frames for a determined time, MRP_RingState in ring open state.

CHK_RC (Check Ring, Ring Closed State)

The MRM shall send its test frames and shall check the link of its ring ports, MRP_RingState in ring closed state..

Local variables of the MRM protocol machine are listed in Table 130.

Table 130 – Local variables of MRM protocol machine

Name	Type	Meaning
SA_Port1	OctetString[6]	Ring port RPort_1 MAC source address of host
SA_Port2	OctetString[6]	Ring port RPort_2 MAC source address of host
SA_RPort	OctetString[6]	Ring port 1 or ring port 2 MAC source address
PRIORITY	Unsigned8	Priority according to IEEE 802.1Q for MRP-PDU. Shall be set to ORG.
MRP_TS_Prio	Unsigned16	MRP_Prio of host
MRP_TS_SA	OctetString[6]	MAC source address of host
RPort_1	Unsigned16	Port identification of ring port 1
RPort_2	Unsigned16	Port identification of ring port 2
PRM_RPort	Unsigned16	Port identification of primary ring port
SEC_RPort	Unsigned16	Port identification of secondary ring port
MRP_MRM_NRmax	Unsigned16	Maximum retransmission count of MRP_TEST-PDU
MRP_MRM_NReturn	Unsigned16	Counter, Range MRP_MRM_NRmax ... 0
TC_NReturn	Unsigned16	Counter, Range MRP_TOPNRmax ... 0
AddTest	Boolean	Send additional MRP-PDU of type MRP_Test after MRP_TSTshortT intervall if TRUE
MRP_LNK_UP	Unsigned16	Constant value to indicate Link Up
MRP_LNK_DOWN	Unsigned16	Constant value to indicate Link Down
NoTC	Boolean	Suppress TopologyChange Requests while in line topology (NoTC = TRUE)

The MRM state machine shall be according to Table 131.

Table 131 – MRM state machine

#	Current State	Event /Condition =>Action	Next State
1	Power On	=> INIT_FDB ADD_MAC_FDB({local},{MC_TEST, MC_CONTROL},ORG) PRM_RPort := RPort_1 SEC_RPort := RPort_2 MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 AddTest := FALSE Set_Port_State.req (PRM_RPort, BLOCKED) Set_Port_State.req (SEC_RPort, BLOCKED)	AC_STAT1
2	AC_STAT1	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => Set_Port_State.req (PRM_RPort, FORWARDING) TestRingReq(MRP_TSTdefaultT)	PRM_UP
3	AC_STAT1	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	AC_STAT1
4	AC_STAT1	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => SEC_RPort := PRM_RPort PRM_RPort := RPort Set_Port_State.req (PRM_RPort, FORWARDING) TestRingReq(MRP_TSTdefaultT)	PRM_UP
5	AC_STAT1	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	AC_STAT1
6	AC_STAT1	TestTimer expired => ignore	AC_STAT1

#	Current State	Event /Condition =>Action	Next State
7	PRM_UP	TestTimer expired => AddTest := FALSE TestRingReq(MRP_TSTdefaultT)	PRM_UP
8	PRM_UP	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	PRM_UP
9	PRM_UP	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => TestTimer.stop SetPortState.req(PRM_RPort, BLOCKED)	AC_STAT1
10	PRM_UP	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	PRM_UP
11	PRM_UP	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 NoTC := TRUE TestRingReq(MRP_TSTdefaultT)	CHK_RC
12	PRM_UP	TestRingInd(MRP_SA, MRP_Prio) /MRP_SA == MRP_TS_SA => MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 NoTC := FALSE TestRingReq(MRP_TSTdefaultT)	CHK_RC
13	PRM_UP	TestRingInd(MRP_SA, MRP_Prio) /MRP_SA != MRP_TS_SA => ignore	PRM_UP
14	PRM_UP	LinkChangeInd(PortMode, Link_status) /!AddTest => AddTest := TRUE TestRingReq(MRP_TSTshortT)	PRM_UP

#	Current State	Event /Condition =>Action	Next State
15	PRM_UP	LinkChangeInd(PortMode, Link_status) /AddTest => ignore	PRM_UP
16	PRM_UP	TopologyChangeInd(MRP_SA, t) => ignore	PRM_UP
17	CHK_RO	TestTimer expired => AddTest := FALSE TestRingReq(MRP_TSTdefaultT)	CHK_RO
18	CHK_RO	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	CHK_RO
19	CHK_RO	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN && !NoTC => PRM_RPort := SEC_RPort SEC_RPort := RPort TestRingReq(MRP_TSTdefaultT) TopologyChangeReq(MRP_ZOPchgT) SetPortState.req(SEC_RPort, BLOCKED)	PRM_UP
20	CHK_RO	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => ignore	CHK_RO
21	CHK_RO	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => SetPortState.req(SEC_RPort, BLOCKED)	PRM_UP
22	CHK_RO	TestRingInd(MRP_SA, MRP_Prio) /MRP_SA == MRP_TS_SA => Set_Port_State.req (SEC_RPort, BLOCKED) MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 NoTC := FALSE TestRingReq(MRP_TSTdefaultT) TopologyChangeReq(MRP_TOPchgT)	CHK_RC

#	Current State	Event /Condition =>Action	Next State
23	CHK_RO	TestRingInd(MRP_SA, MRP_Prio) /MRP_SA != MRP_TS_SA => ignore	CHK_RO
24	CHK_RO	LinkChangeInd(PortMode, Link_status) /!AddTest => AddTest := TRUE TestRingReq(MRP_TSTshortT)	CHK_RO
25	CHK_RO	LinkChangeInd(PortMode, Link_status) /AddTest => Ignore	CHK_RO
26	CHK_RO	TopologyChangeInd(MRP_SA, t) => Ignore	CHK_RO
27	CHK_RC	TestTimer expired /MRP_MRM_NReturn >= MRP_MRM_NRmax && !NoTC => Set_Port_State.req (SEC_RPort, FORWARDING) MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 AddTest := FALSE TopologyChangeReq(MRP_TOPchgT) TestRingReq(MRP_TSTdefaultT)	CHK_RO
28	CHK_RC	TestTimer expired /MRP_MRM_NReturn < MRP_MRM_NRmax => MRP_MRM_NReturn := MRP_MRM_NReturn + 1 AddTest := FALSE TestRingReq(MRP_TSTdefaultT)	CHK_RC
29	CHK_RC	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	CHK_RC
30	CHK_RC	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN && !NoTC => PRM_RPort := SEC_RPort SEC_RPort := RPort Set_Port_State.req (SEC_RPort, BLOCKED) Set_Port_State.req (PRM_RPort, FORWARD) TestRingReq(MRP_TSTdefaultT) TopologyChangeReq(MRP_TOPchgT)	PRM_UP

#	Current State	Event /Condition =>Action	Next State
31	CHK_RC	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => ignore	CHK_RC
32	CHK_RC	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	PRM_UP
33	CHK_RC	TestRingInd(MRP_SA, MRP_Prio) /MRP_SA == MRP_TS_SA => MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 NoTC := FALSE	CHK_RC
34	CHK_RC	TestRingInd(MRP_SA, MRP_Prio) /MRP_SA != MRP_TS_SA => ignore	CHK_RC
35	CHK_RC	LinkChangeInd(PortMode, Link_status) /AddTest => ignore	CHK_RC
36	CHK_RC	LinkChangeInd(PortMode, Link_status) /!AddTest => AddTest := TRUE TestRingReq(MRP_TSTshortT)	CHK_RC
37	CHK_RC	TopologyChangeInd(MRP_SA, t) => ignore	CHK_RC
38	AC_STAT1	LinkChangeInd(PortMode, Link_status) => ignore	AC_STAT1
39	CHK_RO	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN && NoTC => PRM_RPort := SEC_RPort SEC_RPort := RPort TestRingReq(MRP_TSTdefaultT) SetPortState.req(SEC_RPort, BLOCKED)	PRM_UP

#	Current State	Event /Condition =>Action	Next State
40	CHK_RC	TestTimer expired /MRP_MRM_NReturn >= MRP_MRM_NRmax && NoTC => Set_Port_State.req (SEC_RPort, FORWARDING) MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 AddTest := FALSE TestRingReq(MRP_TSTdefaultT)	CHK_RO
41	CHK_RC	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN && NoTC => PRM_RPort := SEC_RPort SEC_RPort := RPort Set_Port_State.req (SEC_RPort, BLOCKED) Set_Port_State.req (PRM_RPort, FORWARD) TestRingReq(MRP_TSTdefaultT)	PRM_UP

4.5.2.2 MRC Protocol Machine for MRP

Figure 38 shows the principal behavior of the protocol machine for a media redundancy client MRC.

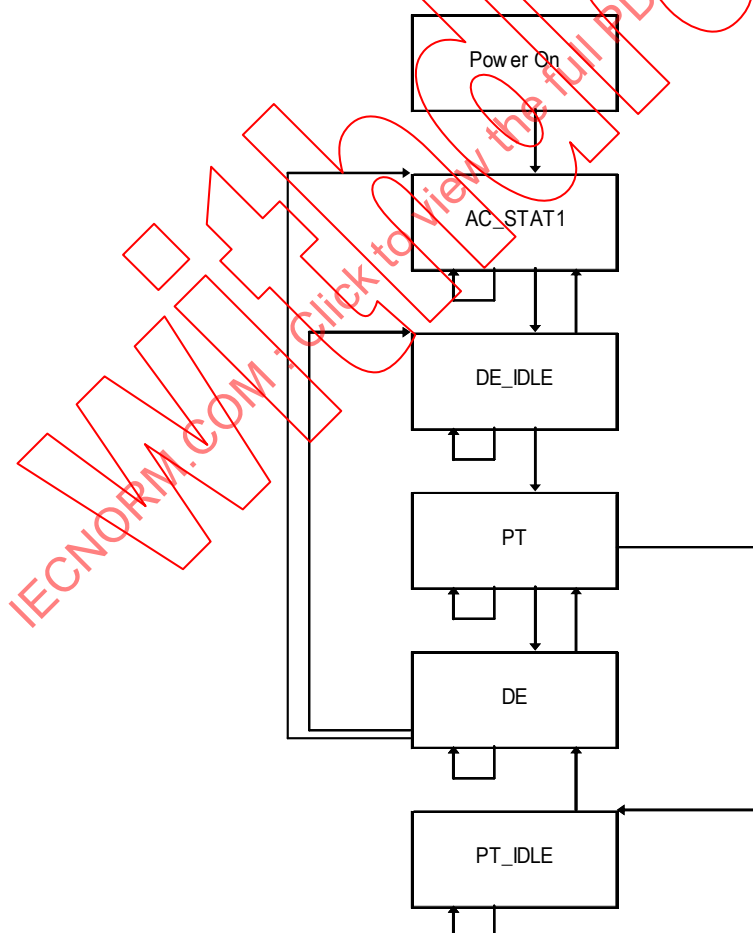


Figure 38 – MRC protocol machine

The protocol machine states perform in general the following tasks:

Power On

Initialization, the MRC shall start with both ring ports RPort_1 and RPort_2 in the port state BLOCKED. Static FDB entries for MRP multicast addresses MC_TEST and MC_CONTROL are generated: Forward MRP frames to MC_TEST and MC_CONTROL between ring ports and frames to MC_CONTROL also to host. All MRP-PDU have highest priority (ORG).

AC_STAT1

Startup, wait for Link Up on one of the ring ports.

DE_IDLE (Data Exchange idle state)

This state shall be reached if only one ring port (primary) has a link and is set to FORWARDING.

PT (Pass Through)

Temporary state while signalling link changes.

DE (Data Exchange)

Temporary state while signalling link changes.

PT_IDLE (Pass Through idle state)

This state shall be reached if both ring ports have a link and are set to FORWARDING.

Local variables of the MRC protocol machine are listed in Table 132.

Table 132 – Local variables of MRC protocol machine

Name	Type	Meaning
SA_RPort	OctetString[6]	Ring port 1 or ring port 2 MAC source address
PRIORITY	Unsigned8	Priority according to IEEE 802.1Q for MRP-PDU. Shall be set to ORG.
RPort_1	Unsigned16	Port identification of ring port 1
RPort_2	Unsigned16	Port identification of ring port 2
PRM_RPort	Unsigned16	Port identification of primary ring port
SEC_RPort	Unsigned16	Port identification of secondary ring port
MRP_LNKNRreturn	Unsigned16	Counter, Range MRP_LNKNRmax ... 0
MRP_LNK_UP	Unsigned16	Constant value to indicate Link Up
MRP_LNK_DOWN	Unsigned16	Constant value to indicate Link Down

The MRC state machine shall be according to Table 133.

Table 133 – MRC state machine

#	Current State	Event /Condition =>Action	Next State
1	Power On	=> INIT_FDB ADD_MAC_FDB({RPort_1,RPort_2}, {MC_TEST, MC_CONTROL},ORG) ADD_MAC_FDB({local},{MC_CONTROL},ORG) PRM_RPort := RPort_1 SEC_RPort := RPort_2 Set_Port_State.req (PRM_RPort, BLOCKED) Set_Port_State.req (SEC_RPort, BLOCKED) UpTimer.ini DownTimer.ini	AC_STAT1
2	AC_STA T1	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => Set_Port_State.req (PRM_RPort, FORWARDING)	DE_IDLE
3	AC_STA T1	MAUType_Change.ind (RPort, Link_status) / Link_status ==MRP_LNK_DOWN => ignore	AC_STAT1
4	AC_STA T1	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => SEC_RPort := PRM_RPort PRM_RPort := RPort Set_Port_State.req (PRM_RPort, FORWARDING)	DE_IDLE
5	AC_STA T1	TopologyChangeInd (MRP_SA, t) => ignore	AC_STAT1
6	DE_IDLE	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => MRP_LNKNReturn := MRP_LNKNRmax UpTimer.start(MRP_LNKupT) LinkChangeReq(PRM_RPort, MRP_LNK_UP, MRP_LNKNReturn * MRP_LNKupT)	PT

#	Current State	Event /Condition =>Action	Next State
7	DE_IDLE	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	DE_IDLE
8	DE_IDLE	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => Set_Port_State.req (PRM_RPort, BLOCKED)	AC_STAT1
9	DE_IDLE	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	DE_IDLE
10	DE_IDLE	TopologyChangeInd(MRP_SA, t) => CLEAR_FDB(t)	DE_IDLE
11	PT	UpTimer expired /MRP_LNKNReturn == 0 => MRP_LNKNReturn := MRP_LNKNRmax Set_Port_State.req (SEC_RPort, FORWARDING)	PT_IDLE
12	PT	UpTimer expired /MRP_LNKNReturn > 0 => MRP_LNKNReturn := MRP_LNKNReturn -1 UpTimer.start(MRP_LNKupT) LinkChangeReq(PRM_RPort, MRP_LNK_UP, MRP_LNKNReturn * MRP_LNKupT)	PT
13	PT	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => ignore	PT

#	Current State	Event /Condition =>Action	Next State
14	PT	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKNReturn := MRP_LNKNRmax UpTimer.stop Set_Port_State.req (SEC_RPort, BLOCKED) DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort , MRP_LNK_DOWN, MRP_LNKNReturn * MRP_LNKdownT)	DE
15	PT	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKNReturn := MRP_LNKNRmax UpTimer.stop PRM_RPort := SEC_RPort SEC_RPort := RPort Set_Port_State.req (PRM_RPort, FORWARDING) Set_Port_State.req (SEC_RPort, BLOCKED) DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort, MRP_LNK_DOWN, MRP_LNKNReturn * MRP_LNKdownT)	DE
16	PT	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore.	PT
17	PT	TopologyChangeInd(MRP_SA, t) => MRP_LNKNReturn := MRP_LNKNRmax UpTimer.stop Set_Port_State.req (SEC_RPort, FORWARDING) CLEAR_FDB(t)	PT_IDLE
18	DE	DownTimer expired /MRP_LNKNReturn == 0 => MRP_LNKNReturn := MRP_LNKNRmax	DE_IDLE

#	Current State	Event /Condition =>Action	Next State
19	DE	DownTimer expired /MRP_LNKNReturn > 0 => MRP_LNKNReturn := MRP_LNKNReturn – 1 DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort, MRP_LNK_DOWN, MRP_LNKNReturn * MRP_LNKdownT)	DE
20	DE	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => MRP_LNKNReturn := MRP_LNKNRmax DownTimer.stop UpTimer.start(MRP_LNKupT) LinkChangeReq(PRM_RPort, MRP_LNK_UP, MRP_LNKNReturn * MRP_LNKupT)	PT
21	DE	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	DE
22	DE	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKNReturn := MRP_LNKNRmax Set_Port_State.req (PRM_RPort, BLOCKED) DownTimer.stop	AC_STAT1
23	DE	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	DE
24	DE	TopologyChangeInd(MRP_SA, t) => MRP_LNKNReturn := MRP_LNKNRmax DownTimer.stop CLEAR_FDB(t)	DE_IDLE

#	Current State	Event /Condition =>Action	Next State
25	PT_IDLE	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => ignore	PT_IDLE
26	PT_IDLE	MAUType_Change.ind (RPort, Link_status) /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKReturn := MRP_LNKmax Set_Port_State.req (SEC_RPort, BLOCKED) DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort , MRP_LNK_DOWN, MRP_LNKReturn * MRP_LNKdownT)	DE
27	PT_IDLE	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKReturn := MRP_LNKmax PRM_RPort := SEC_RPort SEC_RPort := RPort Set_Port_State.req (SEC_RPort, BLOCKED) DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort, MRP_LNK_DOWN, MRP_LNKReturn * MRP_LNKdownT)	DE
28	PT_IDLE	MAUType_Change.ind (RPort, Link_status) /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	PT_IDLE
29	PT_IDLE	TopologyChangeInd(MRP_SA, t) => CLEAR_FDB(t)	PT_IDLE

4.5.2.3 MRM and MRC Functions for MRP

The functions are defined in Table 134.

Table 134 – Functions

Function name	Operations
TestRingReq(t)	SETUP_TEST_RING_REQ TestTimer.start(t)
SETUP_TEST_RING_REQ	Create MRP-PDU according MRP_Test Assignments: MRP_Type := MRP_Test MRP_Prio := MRP_TS_Prio MRP_SA := MRP_TS_SA MRP_PortRole := ring port role of the used port MRP_RingState := actual ring state MRP_Type := MRP_Common MRP_SequenceID := next SequenceID MRP_DomainUUID := assigned domain UUID MRP_Type := MRP_END LMPM_N_Data.req (RPort_1, MC_TEST, SA_Port1, PRIORITY, LT, MRP-PDU) LMPM_N_Data.req (RPort_2, MC_TEST, SA_Port2, PRIORITY, LT, MRP-PDU)
TestRingInd(MRP_SA, MRP_Prio)	LMPM_N_Data.ind (S_Port,,DA, SA,,, N_SDU) MRP-PDU := N_SDU Receive MRP-PDU according MRP_Test MRP_SA := MRP_SA from MRP-PDU MRP_Prio := MRP_Prio from MRP-PDU
TopologyChangeReq(time)	SETUP_TOPOLOGY_CHANGE_REQ (MRP_TOPNRmax * time) if time == 0 CLEAR_LOCAL_FDB else TopTimer.start(MRP_TOPchgT)

Function name	Operations
SETUP_TOPOLOGY_CHANGE_REQ (t)	Create MRP-PDU according MRP_TopologyChange Assignments: MRP_Type := MRP_TopologyChange MRP_Prio := MRP_TS_Prio MRP_SA := MRP_TS_SA MRP_Interval := t MRP_Type := MRP_Common MRP_SequenceID := next SequenceID MRP_DomainUUID := assigned domain UUID MRP_Type := MRP_END LMPM_N_Data.req (RPort_1, MC_CONTROL, SA_Port1, PRIORITY, LT, MRP-PDU) LMPM_N_Data.req (RPort_2, MC_CONTROL, SA_Port2, PRIORITY, LT, MRP-PDU)
TopologyChangeInd(MRP_SA, t)	LMPM_N_Data.ind (S_Port, DA, SA, N_SDU) MRP-PDU := N_SDU Receive MRP-PDU according MRP_TopologyChange MRP_SA := MRP_SA from MRP-PDU t := MRP_Interval from MRP-PDU
LinkChangeReq(RPort, LinkStatus, time)	Create MRP-PDU according MRP_LinkUp or MRP_LinkDown Assignments: if LinkStatus ==MRP_LNK_UP MRP_Type := MRP_LinkUp else MRP_Type := MRP_LinkDown MRP_SA := MRP_TS_SA MRP_PortRole := ring port role of the used port MRP_Interval := time MRP_Type := MRP_Common MRP_SequenceID := next SequenceID MRP_DomainUUID := assigned domain UUID MRP_Type := MRP_END LMPM_N_Data.req (RPort, MC_TEST, SA_RPort, PRIORITY, LT, MRP-PDU)

Function name	Operations
LinkChangeInd(PortMode, LinkStatus)	LMPM_N_Data.ind (S_Port,,DA, SA,,, N_SDU) MRP-PDU := N_SDU Receive MRP-PDU according MRP_LinkDown or MRP_LinkUp PortMode := MRP_Blocked from MRP-PDU (shall be set to 1) if MRP_Type == MRP_LinkUp LinkStatus :=MRP_LNK_UP else LinkStatus :=MRP_LNK_DOWN
CLEAR_FDB(Time)	FDBCclearTimer.start(t)
CLEAR_LOCAL_FDB	Function to Clear local FDB
INIT_FDB	Function to initialize Filtering Data Base
ADD_MAC_FDB(Destination, MAC_Address, Prioty)	Function to add MAC-Address in Filtering Data Base
SetPortState.req(RPort, State)	Function to set the port state of a ring port

4.5.2.4 FDB Clear Timer for MRP

The state table is defined in Table 135.

Table 135 – FDB Clear Timer

#	Current State	Event /Condition => Action	Next State
1	Power On	=> FDBCclearTimer.ini	IDLE
2	IDLE	FDBCclearTimer .expired => CLEAR_LOCAL_FDB	IDLE

4.5.2.5 Topology Change Timer for MRP

The state table is defined in Table 136.

Table 136 – Topology Change Timer

#	Current State	Event /Condition =>Action	Next State
1	Power On	=> TopTimer.ini TC_NReturn := MRP_TOPNRmax - 1	IDLE
2	IDLE	TopTimer expired /TC_NReturn > 0 => SETUP_TOPOLOGY_CHANGE_REQ (TC_NReturn * MRP_TOPchgT) TC_NReturn-- TopTimer.start(MRP_TOPchgT)	IDLE
3	IDLE	TopTimer expired /TC_NReturn <= 0 => TC_NReturn := MRP_TOPNRmax - 1 CLEAR_LOCAL_FDB SETUP_TOPOLOGY_CHANGE_REQ(0)	IDLE

4.5.2.6 RRT Relay Protocol Machine (RRT_RELAY)

4.5.2.6.1 Primitive definitions

4.5.2.6.1.1 Primitives exchanged between RRT_RELAY and ASE

4.5.2.6.1.2 Parameters of RRT_RELAY primitives

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

4.5.2.6.2 State machine description

The RRT relay protocol machine describes the handling of RRT frames in bridges.

Figure 39 shows the RRT_RELAY state machine:

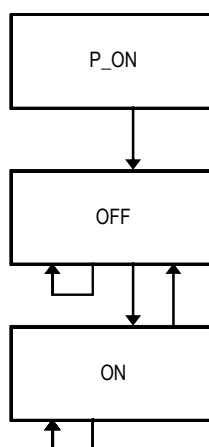


Figure 39 – State transition diagram of RRT_RELAY

States of the RRT_RELAY

P_ON
Initialization
OFF
Deactivated
ON
Activated

4.5.2.6.3 RRT_RELAY state table

Table 137 contains the state table used by RRT_RELAY.

Table 137 – RRT_RELAY state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	=> R_Port1 := NIL R_Port2 := NIL	OFF
2	OFF	RRT_Start.req (RingPort_1, RingPort_2) => R_Port1 := RingPort_1 R_Port2 := RingPort_2 RRT_Start.cnf (OK)	ON
3	OFF	RRT_Stop.req () => R_Port1 := NIL R_Port2 := NIL RRT_Stop.cnf (OK)	OFF
4	OFF	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) => MAC_RelayData.req (CREP, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status)	OFF
5	ON	RRT_Start.req (RingPort_1, RingPort_2) => RRT_Start.cnf (WRONG_SEQUENCE)	ON
6	ON	RRT_Stop.req () => R_Port1 := NIL R_Port2 := NIL RRT_Stop.cnf (OK)	OFF
7	ON	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) /S_Port == LOCAL => for i := R_Port1 to R_Port_2 PUT_IN_LIST_RT_Lo (i)	ON
8	ON	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) /S_Port <> LOCAL && (S_Port <> R_Port1 && S_Port <> R_Port2) => MAC_RelayData.req (CREP, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status)	ON
9	ON	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) /S_Port <> LOCAL && !(S_Port <> R_Port1 && S_Port <> R_Port2) && LOCAL_IS_SOURCE =>	ON
10	ON	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) /S_Port <> LOCAL && !LOCAL_IS_SOURCE && LOCAL_IS_UNIQUE_DESTINATION => RRT_RelayData.ind (CREP, DA, SA, LT, VLANPrio, VLANID, C_SDU, APDU_status)	ON
11	ON	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) /S_Port <> LOCAL&& !(S_Port <> R_Port1 && S_Port <> R_Port2)&& !LOCAL_IS_SOURCE&& !LOCAL_IS_UNIQUE_DESTINATION=>RRT_RelayData.ind (CREP, DA, SA, LT, VLANPrio, VLANID, C_SDU, APDU_status)	ON

#	Current State	Event /Condition =>Action	Next State
12	ON	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) /S_Port <> LOCAL && !(S_Port <> R_Port1 && S_Port <> R_Port2) && !LOCAL_IS_SOURCE && !LOCAL_IS_UNIQUE_DESTINATION && LOCAL_IS_DESTINATION && S_Port == R_Port1 => PUT_IN_LIST_RT_Lo (R_Port2) RRT_RelayData.ind (CREP, DA, SA, LT, VLANPrio, VLANID, C_SDU, APDU_status)	ON
13	ON	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) /S_Port <> LOCAL && !(S_Port <> R_Port1 && S_Port <> R_Port2) && !LOCAL_IS_SOURCE && !LOCAL_IS_UNIQUE_DESTINATION && LOCAL_IS_DESTINATION && S_Port == R_Port2 => PUT_IN_LIST_RT_Lo (R_Port1) RRT_RelayData.ind (CREP, DA, SA, LT, VLANPrio, VLANID, C_SDU, APDU_status)	ON
14	ON	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) /S_Port <> LOCAL && !(S_Port <> R_Port1 && S_Port <> R_Port2) && !LOCAL_IS_SOURCE && !LOCAL_IS_UNIQUE_DESTINATION && !LOCAL_IS_DESTINATION && S_Port == R_Port1 => PUT_IN_LIST_RT_Lo (R_Port2)	ON
15	ON	DEMUX_RRT_RELAY.ind (CREP, S_Port, DA, SA, VLANPrio, VLNAID, C_SDU, APDU_status) /S_Port <> LOCAL && !(S_Port <> R_Port1 && S_Port <> R_Port2) && !LOCAL_IS_SOURCE && !LOCAL_IS_UNIQUE_DESTINATION && !LOCAL_IS_DESTINATION && S_Port == R_Port2 => PUT_IN_LIST_RT_Lo (R_Port1)	ON

4.5.2.6.4 Macros

Table 138 contains the macros used by RRT_RELAY.

Table 138 – Macros used by RRT_RELAY

Name	Function
LOCAL	Reques from local interface (LMPM)
LOCAL_IS_SOURCE	Frame from own device SA == local address
LOCAL_IS_UNIQUE_DESTINATION	Destination address is local address DA == local address
LOCAL_IS_DESTINATION	Destination multicast frame is designed for device
FrameInIngressFilter	The ingress filter criteria fits for this frame. The frame is discarded.
PUT_IN_LIST_RT_Lo (Port)	TX_RT_Lo_List.Insert() TX_RT_Lo_List.Last_Entry.Function := RT_Lo TX_RT_Lo_List.First_Entry.CREP := NIL TX_RT_Lo_List.Last_Entry.S_Port := Port TX_RT_Lo_List.Last_Entry.D_Port := NIL This function enqueue the frame in the corresponding queue of the QueueHandler.

4.5.2.6.5 Functions

No functions defined by the RRT_RELAY.

4.6 Real-time cyclic**4.6.1 FAL syntax description****4.6.1.1 DLPDU abstract syntax reference**

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.6.1.2 RTC APDU abstract syntax

Table 139 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 139 shall represent the content of the DLSDU in Table 4.

Table 139 – RTC APDU syntax

APDU name	APDU structure
RTC-PDU	C_SDU ^ CBA_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 140 defines structures for substitutions of elements of the APDU structures shown in Table 139.

Table 140 – RTC substitutions

Substitution name	Structure
C_SDU	{[DataItem], [GAP*]} * ^a
DataItem	{[IOCS*], [DataObjectElement*]} *
SubstituteDataItem	{[IOCS*] ^b , [SubstituteDataObjectElement*]} *
DataObjectElement	[Data*], IOPS
SubstituteDataObjectElement	[Data*], SubstituteDataValid
CBA_SDU	CBAHeader, CBADataset* ^a
CBAHeader	CBAVersion, CBAFlags, CBACount
CBADataset	CBALength, CBAQualityCode, CBAValue
APDU_Status	CycleCounter, DataStatus, TransferStatus
^a The maximum size of a C_SDU or CBA_SDU shall be not larger than 1 440 octets.	
^b The value shall contain the same value as the field DataItem.IOCS.	

4.6.2 FAL transfer syntax

4.6.2.1 Coding section of Data-RTC-PDU specific fields

4.6.2.1.1 Overview

The field Data and IOxS shall also be used to encode user data for all other PDU types.

4.6.2.1.2 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 contains its local CycleCounter. The receiver of the RTC-PDU or the 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 40.

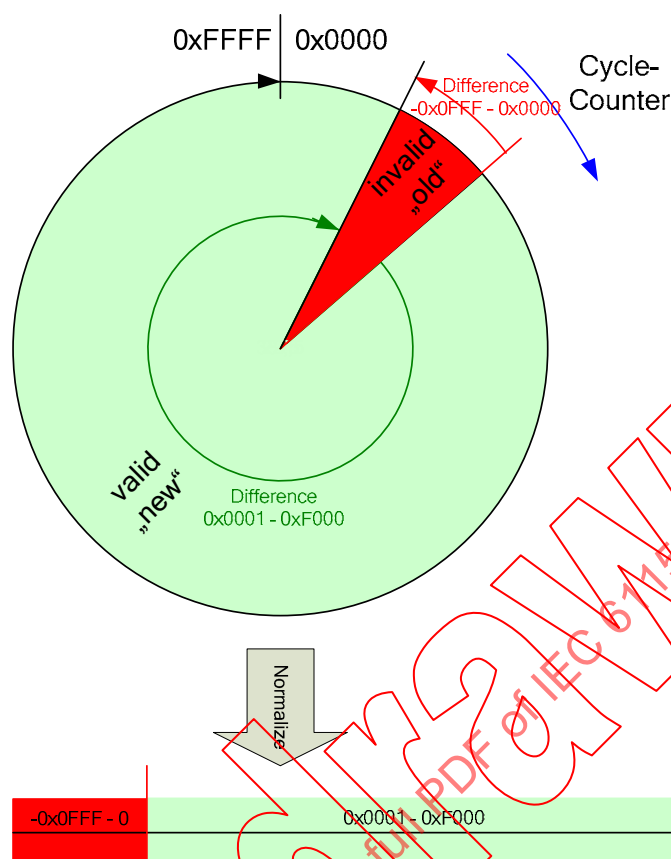


Figure 40 – CycleCounter value range

NOTE The normalization may be done using the following equation:

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

Table 141 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 received RTC-PDU or UDP-RTC-PDU and the value of the current received PDU.

Table 141 – 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 40 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 40 15/16 of the CycleCounter range The consumer votes the received frame as newer than the stored frame	Frame is processed

A device shall maintain a local 64 bit counter with an increment of 31,25 µs as shown in Figure 41.

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. But in case of a global synchronization of the CycleCounter of the RTC-PDUs bit 15 shall be set to avoid a false “older” detection.

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. But in case of a global synchronization of the CycleCounter of the UDP-RTC-PDUs bit 15 shall be set to avoid a false “older” detection.

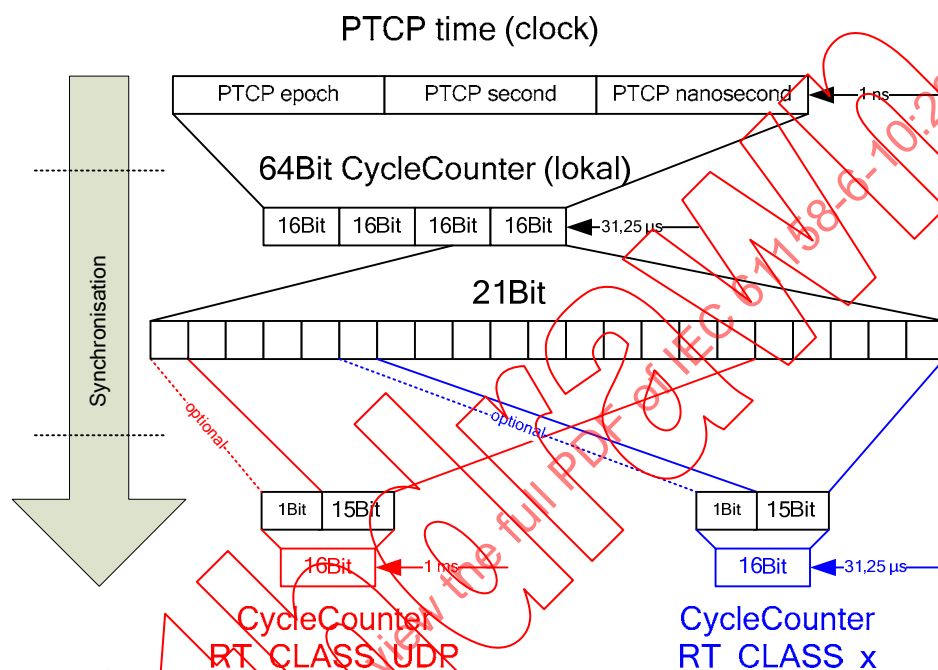


Figure 41 – Structure of the CycleCounter

NOTE If necessary, the local CycleCounter may tightly synchronized global in the network.

In the context of the setting of the CycleCounter the optimized modell shown in Figure 42 shall be used.

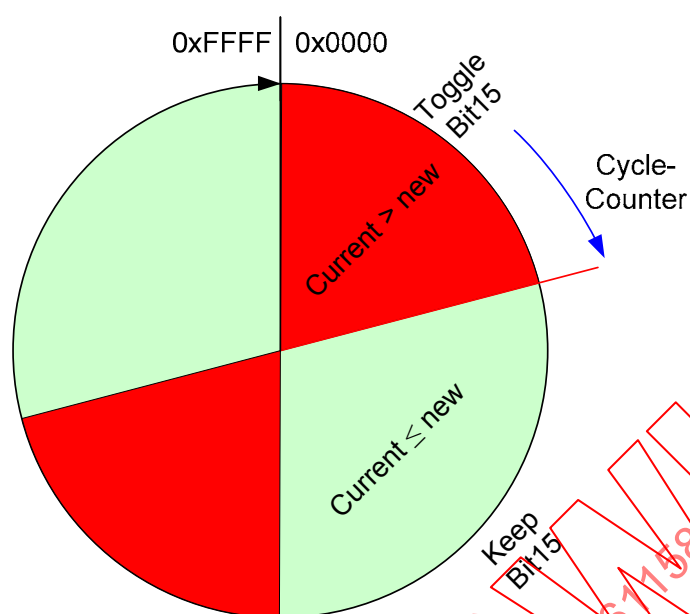


Figure 42 – Optimized CycleCounter setting

if CycleCounter(current)[0..14] <= CycleCounter(New)[0..14]

then /* Current ≤ new */

SET CycleCounter[0..14] with CycleCounter(New)[0..14] and keep CycleCounter[15]

else /* Current > new */

SET CycleCounter[0..14] with CycleCounter(New)[0..14] and toggle CycleCounter[15]

4.6.2.1.3 Coding of the field DataStatus

The coding of this field shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0: DataStatus.State

This field shall be coded with the values according to Table 142.

Table 142 – 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 143.

Table 143 – DataStatus.Redundancy

Value (hexadecimal)	Meaning
0x00	Reserved

Bit 2: DataStatus.DataValid

This field shall be coded with the values according to Table 144.

Table 144 – DataStatus.DataValid

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

Bit 3: DataStatus.reserved_2

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 145.

Table 145 – DataStatus.ProviderState

Value (hexadecimal)	Meaning
0x00	Stop
0x01	Run

Bit 5: DataStatus.StationProblemIndicator

This field shall be coded with the values according to Table 146.

Table 146 – DataStatus.StationProblemIndicator

Value (hexadecimal)	Meaning
0x00	Problem detected
0x01	Normal Operation

Bit 6: DataStatus.reserved_3

This field shall be coded with the values according to Table 147.

Table 147 – DataStatus.reserved_3

Value (hexadecimal)	Meaning
0x00	Reserved

Bit 7: DataStatus.Ignore

This field shall be coded with the values according to Table 148.

Table 148 – DataStatus.Ignore of a frame

Value (hexadecimal)	Meaning
0x00	Evaluate the DataStatus
0x01	Reserved

4.6.2.1.4 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 149.

Table 149 – 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 E.g. 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, e.g. the bridge received a header, but the data have not arrived on time
Other	None	Reserved	Reserved

4.6.2.1.5 Coding section related to decentralized periphery**4.6.2.1.5.1 Coding of the field Data**

One of the following data types shall be used for each Data field:

Boolean
 Integer
 Unsigned
 Floating Point
 Visible String
 OctetString
 Binary Date
 Time of Day
 Time-Difference
 Network Time
 Network Time Difference

4.6.2.1.5.2 Coding section related to IOxS (IOPS, IOCS, SubstituteDataValid)

4.6.2.1.5.2.1 Coding of the field IOPS

The coding of these fields shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0: IOxS.Extension

This field shall be coded with the values according to Table 150.

Table 150 – 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 3.7.3.2.

Bit 5 to 6: IOxS.Instance

This field shall be coded with the values according to Table 151 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 151 – IOCS.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 152.

Table 152 – 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

4.6.2.1.5.2.2 Coding of the field IOCS

The coding of these fields shall be according to 4.6.2.1.5.2.1.

4.6.2.1.5.2.3 Coding of the field SubstituteDataValid

The coding of these fields shall be according to 4.6.2.1.5.2.1.

4.6.2.1.6 Coding section related to distributed automation

4.6.2.1.6.1 Coding of the field CBAValue

One of the following data types shall be used for each CBAValue field:

VARIANT_BOOL
char
unsigned char
short
unsigned short
long
unsigned long
float
double
date
BSTR
SAFEARRAY
struct

4.6.2.1.6.2 Coding of the field CBAVersion

This field shall be coded as data type Unsigned8. This field shall be set to 0x11.

4.6.2.1.6.3 Coding of the field CBAFlags

This field shall be coded as data type Unsigned8. This field shall be set to 0x00.

4.6.2.1.6.4 Coding of the field CBACount

This field shall be coded as data type Unsigned16. This field shall be set to the number of following CBADataset fields.

4.6.2.1.6.5 Coding of the field CBALength

This field shall be coded as data type Unsigned16. This field shall be set to the number of octets of the fields CBALength, CBAQualityCode, and CBAValue.

4.6.2.1.6.6 Coding of the field CBAQualityCode

This field shall be coded as data type ITEMQUALITYDEF.

4.6.3 Application Relationship Protocol Machines (ARPMs)

4.6.3.1 Provider protocol machine (PPM)

4.6.3.1.1 Primitive definitions

4.6.3.1.1.1 Primitives exchanged between PPM and PPM user

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

4.6.3.1.1.2 Primitives exchanged between PPM and LPM

The service primitives including their associated parameters issued by PPM received by LPM and vice versa are described in the LPM ASE in the service definition.

4.6.3.1.1.3 Parameters of PPM primitives

The parameters used with the primitives exchanged between the PPM user and the PPM are described in FAL Service Definition.

4.6.3.1.2 State machine description

The W-START state indicates that an initialisation is needed. The Activate service sets the machine in the RUN state waiting for an Provider Data service requests and Time events. After issuing the transmission of data the CRUN state is used to wait for a confirmation. 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. A Close in CRUN has to wait for the outstanding confirmation in the WCON state before entering W-START.

Local variables of the PPM

Buffer_Data (OctetString)

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 CL-RPC-PDU with NDRDataCyclic.

NOTE DataItem* includes all DataItems for the dedicated PDU.

Buffer_Status (Unsigned 16)

This local variable contains the status (encoded as APDU_Status) which shall be used as storage of the status dedicated for conveyance of the next Data-RTC-PDU or CL-RPC-PDU with NDRDataCyclic.

New_Stat (Boolean)

This local variable will be used to indicate the first successful transmission of data after Open.

4.6.3.1.3 PPM state table

Table 153 contains the complete description of the PPM state table.

Table 153 – PPM state table

#	Current State	Event /Condition =>Action	Next State
1	W-START	PPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, TxOption, ReductionRatio, Phase, Sequence, Default_Values, Default_Status) => New_Stat := FALSE; Buffer_Data := Default_Values Buffer_Status := Default_Status LMPM_Schedule_add.req (CREP, ReductionRatio, Phase, Sequence) PPM_Activate.cnf (+)(CREP)	RUN
2	W-START	PPM_Close.req(CREP) => PPM_Close.cnf (+)(CREP)	W-START
3	W-START	PPM_Set_Prov_Data.req (CREP,Data) => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Prov_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	W-START
4	W-START	PPM_Set_Prov_Status.req (CREP,D_Status) => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Prov_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	W-START
5	W-START	C_Data_cnf => ignore	W-START
6	RUN	PPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, TxOption, ReductionRatio, Phase, Sequence, Default_Values, Default_Status) => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Activate.cnf (-)(CREP,ERRCLS,ERRCODE)	RUN
7	RUN	PPM_Close.req(CREP) => LMPM_Schedule_remove.req(CREP) PPM_Close.cnf (+)(CREP)	W-START
8	RUN	PPM_Set_Prov_Data.req (CREP,Data) => Buffer_Data := Data PPM_Set_Prov_Data.cnf(+) (CREP)	RUN
9	RUN	PPM_Set_Prov_Status.req (CREP,D_Status) => Buffer_Status := D_Status PPM_Set_Prov_Status.cnf(+) (CREP)	RUN

#	Current State	Event /Condition =>Action	Next State
10	RUN	C_Data_cnf => ignore	RUN
11	RUN	LMPM_Time_Event.ind(CREP, PortList) /PortList == NIL => C_SDU := Buffer_Data, APDU_Status := (Cycle_Count, Buffer_Status) C_Data_req(PortList)	CRUN
12	RUN	LMPM_Time_Event.ind(CREP, PortList) /PortList !=NIL => C_SDU := Buffer_Data, APDU_Status := (Cycle_Count, Buffer_Status) C_Data_req(PortList)	CRUN
13	CRUN	PPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, TxOption, ReductionRatio, Phase, Sequence, Default_Values, Default_Status) => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Activate.cnf (-)(CREP,ERRCLS,ERRCODE)	CRUN
14	CRUN	PPM_Close.req(CREP) => LMPM_Schedule_remove.req(CREP) PPM_Close.cnf (+)(CREP)	WCON
15	CRUN	C_Data_cnf /LMPM_status != LS/IV && !New_Stat => New_Stat := TRUE PPM_Start.ind(CREP)	RUN
16	CRUN	C_Data_cnf /LMPM_status != LS/IV && New_Stat =>	RUN
17	CRUN	C_Data_cnf /LMPM_status == LS/IV => PPM_Error.ind(CREP,No_Data_Send)	RUN
18	CRUN	LMPM_Time_Event.ind(CREP, PortList) => PPM_Error.ind(CREP,No_Data_Send)	CRUN

#	Current State	Event /Condition =>Action	Next State
19	CRUN	PPM_Set_Prov_Data.req (CREP,Data) => Buffer_Data := Data PPM_Set_Prov_Data.cnf(+) (CREP)	CRUN
20	CRUN	PPM_Set_Prov_Status.req (CREP,D_Status) => Buffer_Status := D_Status PPM_Set_Prov_Status.cnf(+) (CREP)	CRUN
21	WCON	PPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, TxOption, ReductionRatio, Phase, Sequence, Default_Values, Default_Status) => New_Stat := FALSE; Buffer_Data := Default_Values Buffer_Status := Default_Status LMPM_Schedule_add.req (CREP, ReductionRatio, Phase, Sequence) PPM_Activate.cnf (+)(CREP)	CRUN
22	WCON	PPM_Close.req(CREP) => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Close.cnf (-)(CREP,ERRCLS,ERRCODE)	WCON
23	WCON	C_Data_cnf /LMPM_status != LS/IV =>	W-START
24	WCON	C_Data_cnf /LMPM_status == LS/IV => PPM_Error.ind(CREP,No_Data_Send)	W-START
25	WCON	LMPM_Time_Event.ind(CREP, PortList) => PPM_Error.ind(CREP,No_Data_Send)	WCON
26	WCON	PPM_Set_Prov_Data.req (CREP,Data) => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Prov_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	WCON
27	WCON	PPM_Set_Prov_Status.req (CREP,D_Status) => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Prov_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	WCON

4.6.3.1.4 Functions

Table 154 contains the functions or macros used by the PPM, their arguments and their descriptions.

Table 154 – Functions used by the PPM

Function name	Description
C_Data_cnf	if (txOption == RTC) LMPM_C_Data.cnf (CREP, LMPM_status, Cycle) if (txOption == UDP) UDP_C_Data.cnf (IPPort, CREP, LMPM_status, Cycle)
C_Data_req(PortList)	if (txOption == RTC) LMPM_C_Data.req (CREP, PortList, DA, SA, Prio, C_SDU, APDU_Status) if (txOption == UDP) UDP_C_Data.req (IPPort, CREP, DA, SA, Prio, C_SDU, APDU_Status)

4.6.3.2 Consumer protocol machine (CPM)

4.6.3.2.1 Primitive definitions

4.6.3.2.1.1 Primitives exchanged between CPM and CPM user

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

4.6.3.2.1.2 Primitives exchanged between CPM and LMPM

The service primitives including their associated parameters issued by CPM received by LMPM and vice versa are described in the LMPM ASE in the service definition.

4.6.3.2.1.3 Parameters of CPM primitives

The parameters used with the primitives exchanged between the CPM user and the CPM are described in FAL Service Definition.

4.6.3.2.2 State machine description

The W-START state indicates that the initialisation is needed. The Activate service sets the machine in the RUN state or in the FRUN state. The state machine is waiting for the first LMPM_C_Data service indication in these states. After passing the Start indication to the user the RUN state is entered. A timeout will force a state transition back to FRUN (combined with a Stop indication). A Close service request is needed to reenter the W-START state.

Each time receiving valid C_Data.ind the time is read and stored in a local variable together with a Flag.

Local variables of the CPM

Cycle (Unsigned 16)

This local variable contains the last valid Cycle Counter conveyed with a LMPM_C_Data indication with valid data.

RecvCnt (Unsigned 16)

This local variable contains the counter value of the received cyclic PDU since last Cons Status request.

Buffer_Data (Octet String)

This local variable contains the last valid service data unit conveyed with a LMPM_C_Data indication.

Buffer_Status (Unsigned 16)

This local variable contains the last valid APDU_Status conveyed with a LMPM_C_Data indication.

New_Data (Boolean)

This local variable signals the receipt of a valid LMPM_C_Data indication (set to FALSE by a Get_Cons_Data request).

WDt (Unsigned 16)

This local variable contains the counter value of the time events since last LMPM_C_Data indication.

DHt (Unsigned 16)

This local variable contains the counter value of the time events since last LMPM_C_Data indication with valid data.

4.6.3.2.3 CPM state table

Table 155 contains the complete description of the CPM state table.

Withdrawing
IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 155 – CPM state table

#	Current State	Event /Condition =>Action	Next State
1	W-START	CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Length, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status) /StartTon = TRUE => InitCy := TRUE Store DA,SA,FrameID,Prio, VLAN, Timeout_Base, WatchDogFactor, DataHoldFactor, Default_Value, Default_Status New_Data := FALSE, Intime := FALSE, for all ports of interface do: RxA(Port) , RxP(Port) := FALSE WDt := 0, Dht := 0 Cycle := 0, RecvCnt := 0, Buffer_Data, AuxBuffer_Data:= Default_Value, Buffer_Status, AuxBuffer_Status := Default_Status Primary := TRUE LMPM_Schedule_add.req (CREP, ReductionRatio:=Timeout_Base, Phase:=0, Sequence:=0) CPM_Activate.cnf (+)(CREP)	RUN
2	W-START	CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Length, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status) /StartTon = FALSE => Store DA,SA,FrameID,Prio, VLAN, Timeout_Base, WatchDogFactor, DataHoldFactor, Default_Value, Default_Status New_Data := FALSE, Intime := FALSE, for all ports of interface do: RxA(Port) , RxP(Port) = FALSE WDt := 0, Dht := 0 Cycle := 0, RecvCnt := 0, Buffer_Data, AuxBuffer_Data:= Default_Value, Buffer_Status, AuxBuffer_Status := Default_Status Primary := TRUE LMPM_Schedule_add.req (CREP, ReductionRatio:=Timeout_Base, Phase:=0, Sequence:=0) CPM_Activate.cnf (+)(CREP)	FRUN
3	W-START	CPM_Close.req(CREP) => CPM_Close.cnf (+)(CREP)	W-START
4	W-START	CPM_Get_Cons_Status.req (CREP) => ERRCLS := CXTX ERRCODE := INVALID_STATE CPM_Get_Cons_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	W-START
5	W-START	CPM_Set_RedRole.req (CREP, RedRole) => Primary := RedRole CPM_Set_RedRole.cnf (CREP)	W-START
6	W-START	CPM_Get_Cons_Data.req (CREP) => ERRCLS := CXTX ERRCODE := INVALID_STATE CPM_Get_Cons_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	W-START
7	W-START	C_Data_ind => ignore	W-START

#	Current State	Event /Condition =>Action	Next State
8	FRUN	CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Length, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status) => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Activate.cnf (-) (CREP,ERRCLS,ERRCODE)	FRUN
9	FRUN	CPM_Close.req(CREP) => LMPM_Schedule_remove.req(CREP) CPM_Close.cnf (+)(CREP)	W-START
10	FRUN	CPM_Get_Cons_Status.req (CREP) => Status := Cycle, Buffer_Status, RecvCounter := RecvCnt, RecvCnt := 0 CPM_Get_Cons_Status.cnf(+) (CREP, Status, RecvCounter)	FRUN
11	FRUN	CPM_Set_RedRole.req (CREP, RedRole) => Primary := RedRole CPM_Set_RedRole.cnf (CREP)	FRUN
12	FRUN	CPM_Get_Cons_Data.req (CREP) => Data := Buffer_Data, New_Flag := New_Data, New_Data := FALSE CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)	FRUN
13	FRUN	C_Data_ind /APDU_Status.DataStatus.DataValid && CHECK_PAR_C => InitCy := FALSE, New_Data := TRUE, RxA(Port) := TRUE, DHT := 0, WDt := 0, Cycle:=APDU_Status.Cycle_Counter, Buffer_Data := Filter (Data) Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1 CPM_Start.ind(CREP) CPM_New_Cons_Data.ind(CREP, APDU_Status)	RUN
14	FRUN	C_Data_ind / !APDU_Status.DataStatus.DataValid && !Primary && CHECK_PAR_C => InitCy := FALSE, RxA(Port) := TRUE, DHT := 0, WDt := 0, Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1 CPM_Start.ind(CREP)	RUN
15	FRUN	C_Data_ind /!APDU_Status.DataStatus.DataValid && Primary && CHECK_PAR_C => RxA(Port) := TRUE, WDt := 0, Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1	FRUN
16	FRUN	C_Data_ind /!CHECK_PAR_C => ignore	FRUN
17	FRUN	LMPM_Path_Check_Event.ind(CREP) /for all Port RxA(Port) == RxP(Port) => ignore	FRUN

#	Current State	Event /Condition =>Action	Next State
18	FRUN	LMPM_Path_Check_Event.ind(CREP) //(for all Port RxA(Port) == RxP(Port)) => for all Port RxC(Port) := RxP(Port) XOR RxA(Port) RxP(Port) := RxA(Port) CPM_Path_Change(list of RxC, list of RxA)	FRUN
19	RUN	CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Length, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status) => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Activate.cnf (-) (CREP,ERRCLS,ERRCODE)	RUN
20	RUN	CPM_Close.req(CREP) => LMPM_Schedule_remove.req(CREP) CPM_Close.cnf (+)(CREP)	W-START
21	RUN	CPM_Get_Cons_Status.req (CREP) => Status := Cycle, Buffer_Status, RecvCounter := RecvCnt, RecvCnt := 0 CPM_Get_Cons_Status.cnf(+) (CREP, Status, RecvCounter)	RUN
22	RUN	CPM_Set_RedRole.req (CREP, RedRole) => Primary := RedRole CPM_Set_RedRole.cnf (CREP)	RUN
23	RUN	CPM_Get_Cons_Data.req (CREP) => Data := Buffer_Data, New_Flag := New_Data, New_Data := FALSE CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)	RUN
24	RUN	C_Data_ind //(APDU_Status.CycleCounter-Cycle)>> InitCy) && (APDU_Status.DataStatus.DataValid) && CHECK_PAR_C && INTIME => Intime := TRUE, Intm := TRUE, InitCy := FALSE, New_Data := TRUE, RxA(Port) := TRUE, DHt := 0, WDt := 0, Cycle:= APDU_Status.Cycle_Counter, Buffer_Data := Filter (Data) Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1 CPM_New_Cons_Data.ind(CREP, APDU_Status, Intm)	RUN
25	RUN	C_Data_ind //(APDU_Status.CycleCounter-Cycle)>> InitCy) && (APDU_Status.DataStatus.DataValid) && CHECK_PAR_C && !INTIME && !Intime => InitCy := FALSE, New_Data := TRUE, RxA(Port) := TRUE, DHt := 0, WDt := 0, Cycle:= APDU_Status.Cycle_Counter, AuxBuffer_Data := Filter (Data) AuxBuffer_Status := APDU_Status RecvCnt := RecvCnt + 1 CPM_New_Cons_Data.ind(CREP, APDU_Status, Intm)	RUN

#	Current State	Event /Condition =>Action	Next State
26	RUN	C_Data_ind /((APDU_Status.CycleCounter-Cycle)>> InitCy) && (APDU_Status.DataStatus.DataValid) && CHECK_PAR_C && !INTIME && Intime => ignore	RUN
27	RUN	C_Data_ind /((APDU_Status.CycleCounter-Cycle)>> InitCy) && !APDU_Status.DataStatus.DataValid && !Primary && CHECK_PAR_C => InitCy := FALSE, RxA(Port) := TRUE, DHt := 0, WDt := 0, Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1	RUN
28	RUN	C_Data_ind /((APDU_Status.CycleCounter-Cycle)>> InitCy) && (!APDU_Status.DataStatus.DataValid && Primary) && CHECK_PAR_C => InitCy := FALSE, RxA(Port) := TRUE, WDt := 0, Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1	RUN
29	RUN	C_Data_ind /((APDU_Status.CycleCounter-Cycle)<< && !InitCy) && CHECK_PAR_C => RxA(Port) := TRUE	RUN
30	RUN	C_Data_ind /!CHECK_PAR_C => ignore	RUN
31	RUN	LMPM_Time_Event.ind(CREP) /DHt > (DataHoldTime-1) && WDt > (WatchDogTime-1) => Intime := FALSE DHt := 0, WDt := 0 CPM_NoData.ind(CREP) CPM_Stop.ind(CREP)	FRUN
32	RUN	LMPM_Time_Event.ind(CREP) /DHt > (DataHoldTime-1) && WDt < (WatchDogTime) => DHt := 0, WDt := WDt+1 CPM_Stop.ind(CREP)	FRUN
33	RUN	LMPM_Time_Event.ind(CREP) /DHt < (DataHoldTime) && WDt > (WatchDogTime-1) => Intime := FALSE DHt := DHt +1, WDt := 0 CPM_NoData.ind(CREP)	RUN
34	RUN	LMPM_Time_Event.ind(CREP) /DHt < (DataHoldTime) && WDt < (WatchDogTime) => DHt := DHt +1, WDt := WDt +1	RUN
35	RUN	LMPM_Path_Check_Event.ind(CREP) /for all Port RxA(Port) == RxP(Port) => ignore	RUN

#	Current State	Event /Condition =>Action	Next State
36	RUN	LMPM_Path_Check_Event.ind(CREP) /(for all Port RxA(Port) == RxP(Port)) => for all Port RxC(Port) := RxP(Port) XOR RxA(Port) RxP(Port) := RxA(Port) CPM_Path_Change(list of RxC, list of RxA)	RUN

4.6.3.2.4 Functions

Table 156 contains the functions or macros used by the CPM, their arguments and their descriptions.

Table 156 – Functions used by the CPM

Function name	Description
C_Data_ind	if (txOption == RTC) LMPM_C_Data.ind (CREP, Port, DA, SA, Prio, C_SDU, APDU_Status) if (txOption == UDP) UDP_C_Data.ind (IPPort, CREP, DA, SA, Prio, C_SDU, APDU_Status)
<<	Older Result of the subtraction of two 16 bit unsigned integers is between -0xffff and 0 Sign overflows will not be calculated
>>	Newer Means not << (result between 0x8000 and 0x1000 or between 1 and 0x7FFF)
CHECK_PAR_C	C_SDU.Length == Exp_Length && SA == stored SA && APDU_Status.TransferStatus == 0 && if (txOption == UDP) IPPort == Exp_IPPort
INTIME	OperationMode != Sync Data received in Red interval

4.7 Real-time acyclic

4.7.1 RTA 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 RTA APDU abstract syntax

Table 157 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 157 shall represent the content of the DLSDU in Table 4.

Table 157 – 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

Table 158 defines structures for substitutions of elements of the APDU structures shown in Table 157.

Table 158 – RTA substitutions

Substitution name	Structure
Reference	AlarmDstEndpoint, AlarmSrcEndpoint
FlagsSequence	AddFlags, SendSeqNum, AckSeqNum
DATA-RTA-PDU	Reference, PDUType (=1), FlagsSequence, VarPartLen (1 – 1 432) ^a , RTA-SDU, RTAPadding* ^b
ACK-RTA-PDU	Reference, PDUType (=3), FlagsSequence, VarPartLen (=0), RTAPadding* ^c
NACK-RTA-PDU	Reference, PDUType (=2), FlagsSequence, VarPartLen (=0), RTAPadding* ^d
ERR-RTA-PDU	Reference, PDUType (=4), FlagsSequence, VarPartLen (=4), PNIOStatus, RTAPadding* ^e
^a The maximum VarPartLen is calculated to fit in the RTA-PDU and in the UDP-RTA-PDU. ^b The number of padding octets shall be in the range of 0..31 depending on the RTA-SDU size ^c The number of padding octets shall be 32 for RTA_CLASS_1 and shall be 4 for RTA_CLASS_UDP ^d The number of padding octets shall be 32 for RTA_CLASS_1 and shall be 4 for RTA_CLASS_UDP ^e The number of padding octets shall be 28 for RTA_CLASS_1 and shall be 0 for RTA_CLASS_UDP	

4.7.2 RTA transfer syntax

4.7.2.1 Coding section related to RTA-PDUs specific fields

4.7.2.1.1 Coding of the field AlarmDstEndpoint

This field shall be coded as data type Unsigned16.

4.7.2.1.2 Coding of the field AlarmSrcEndpoint

This field shall be coded as data type Unsigned16.

NOTE The value of the field reference for destination is provided by the receiver. The value of the field reference for the source is provided by the sender. The fields are specific for a connection and exchanged during context management operation.

4.7.2.1.3 Coding of the field PDUType

The coding of this field shall be according to 3.7.3.3 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 159.

Table 159 – PDType.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: PDType.Version

This field shall be coded with the values according to Table 160.

Table 160 – PDType.Version

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Version 1 of the protocol
0x02 – 0x0F	Reserved

4.7.2.1.4 Coding of the field AddFlags

The coding of this field shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0 – 3: AddFlags.WindowSize

This field shall be set to one.

Bit 4: AddFlags.TACK

This field shall be set by the sender within a RTA-PDU to control the acknowledge behavior of the receiver.

The value one shall be set for immediate acknowledge in a DATA-RTA-PDU.

The value zero shall be set for ERR-RTA-PDU, ACK-RTA-PDU, and NACK-RTA-PDU but the receiver shall not check it.

Bit 5 – 7: AddFlags.reserved

This field shall be set according to 3.7.3.2.

4.7.2.1.5 Coding of the field SendSeqNum

This field shall be coded as data type Unsigned16 with the following values:

Hexadecimal (0x0000 – 0x7FFF)

contains a valid SendSeqNum of a DATA-RTA-PDU

This field contains the number of the DATA-RTA-PDU. The first issued DATA-RTA-PDU shall contain the SendSeqNum 0. The incrementing and comparison of this number is done using modulo 2^{15} operations.

Hexadecimal (0xFFFFE, 0xFFFF)

These values are used to synchronize sender and receiver after establishment of the application relationship. The value 0xFFFF indicates the first DATA-RTA-PDU. The value 0xFFFFE indicates that there was no reception of a DATA-RTA-PDU before.

NOTE The first DATA-RTA-PDU sets SendSeqNum=0xFFFF and AckSeqNum=0xFFFE. It is acknowledged with SendSeqNum=0xFFFE and AckSeqNum=0xFFFF. The second DATA-RTA-PDU sets SendSeqNum=0 and AckSeqNum=0xFFFE. The synchronization is necessary because the acyclic protocol does not define any connection monitoring.

4.7.2.1.6 Coding of the field AckSeqNum

This field shall be coded as data type Unsigned16 with the following values:

Hexadecimal (0x0000 – 0x7FFF)

contains a valid AckSeqNum

This field contains the number of the DATA-RTA-PDU that is expected to be acknowledged or which is acknowledged.

Hexadecimal (0xFFFF, 0xFFFE)

contains an initial AckSeqNum

The value 0xFFFE indicates that there was no DATA-RTA-PDU received before. The value 0xFFFF indicates acknowledges the reception of the first DATA-RTA-PDU.

4.7.2.1.7 Coding of the field VarPartLen

This field shall be coded as data type Unsigned16 according to Table 161. The VarPartLen shall not contain the RTAPadding.

Table 161 – VarPartLen

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

4.7.3 Application Relationship Protocol Machines (ARPMs)

4.7.3.1 APMS

4.7.3.1.1 Primitive definitions

4.7.3.1.1.1 Primitives exchanged between APMS and APMS user

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

4.7.3.1.1.2 Primitives exchanged between APMS and LMPM

The service primitives including their associated parameters issued by APMS received by LMPM and vice versa are described in the Common DL Mapping ASE in the service definition.

4.7.3.1.1.3 Parameters of APMS primitives

The parameters used with the primitives exchanged between the APMS user and the APMS are described in FAL Service Definition.

4.7.3.1.2 State machine description

The CLOSED state indicates that an initialisation is needed. Activate service sets the machine in the OPEN state waiting for an A-Data service request. After issuing the transmission of data the WACK state is used to wait for an acknowledge. Receiving the acknowledge will set

back the machine to the OPEN state. A Close service request resets the protocol machine to the CLOSED state.

Local variables of the APMS

PDU

A local variable representing the UDP-RTA-PDU or the RTA-PDU depending on the transport. Used to show the protocol specific setup and validation of RTA fields. The setup and validation of additional PDU-Parameters (e.g. IP-header) is not shown.

PDUType: 1 = DATA, 2 = NACK, 3 = ACK, 4 = ERR

Seq_Count (Unsigned 16)

This local variable contains the counter value which shall be used at the conveyance of the next DATA-RTA-PDU.

Seq_CountO (Unsigned 16)

This local variable contains the previous counter value of the Seq_Count variable.

Retry (Unsigned8)

This local variable is loaded with MRetry and decremented at every retransmission of a frame. A value of zero indicates the failure in that transaction. A new Activate service is needed to re-establish the communication between the peer providers.

TimeAct (Boolean)

This local variable will be used to find the first full timer cycle after transmitting or receiving a PDU.

4.7.3.1.3 APMS state table

Table 162 contains the complete description of the APMS state machine. A LMPM_A_Data.ind primitive will be accepted if the value of the PDUType.Type field is DATA-RTA-PDU, ACK-RTA-PDU or NACK-RTA-PDU and the PDUType.Version is RTA_VERS=1.

The APMS state machine specify 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.

Table 162 – APMS state table

#	Current State	Event /Condition =>Action	Next State
1	CLOSED	APMS_Activate.req(CREP,DA,SA, FrameID,VLANPrio, VLANID, RTATimeoutFactor, MRetry,Transport, DstIP, SrcIP) => Seq_Count:=0xffff Seq_CountO:=0xfffe Store DA,SA,FrameID,VLANPrio, VLAN, M_Retry, Transport, DstIP, SrcIP A_Timer_add.req (CREP,RTATimeoutFactor) APMS_Activate.cnf (+)(CREP)	OPEN
2	CLOSED	APMS_Close.req(CREP) => APMS_Close.cnf (+)(CREP)	CLOSED
3	CLOSED	APMS_A_Data.req(CREP, Data) => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_A_Data.cnf (-)(CREP,ERRCLS,ERRCODE)	CLOSED
4	CLOSED	A_Data_ind => ignore	CLOSED
5	CLOSED	A_Data_cnf => ignore	CLOSED
6	OPEN	APMS_Activate.req(CREP,DA,SA, FrameID,VLANPrio, VLANID, RTATimeoutFactor, MRetry, Transport, DstIP, SrcIP) => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_Activate.cnf (-)(CREP,ERRCLS,ERRCODE)	OPEN
7	OPEN	APMS_Close.req(CREP,ErrCode) /VLANPrio == Low => PDU.Version:=1 PDU.PDUType:=ERR PDU.AddFlags.TACK:=0 PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Seq_Count PDU.AckSeqNum:=Seq_CountO of APMR PDU.PNIOStatus.ErrorCode = 0xCF PDU.PNIOStatus.ErrorDecode = 0x81 PDU.PNIOStatus.ErrorCode1 = 0xFD PDU.PNIOStatus.ErrorCode2 = ErrCode A_Data_req A_Timer_remove.req(CREP) APMS_Close.cnf (+)(CREP)	CLOSED
8	OPEN	APMS_Close.req(CREP) /VLANPrio == High => A_Timer_remove.req(CREP) APMS_Close.cnf (+)(CREP)	CLOSED
9	OPEN	APMS_A_Data.req(CREP, Data) =>PDU.Version:=1 PDU.PDUType:=DATA PDU.AddFlags.TACK:=Tack PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Seq_Count PDU.AckSeqNum:=Seq_CountO of APMR PDU.RTA-SDU:=Data Store PDU.Flags, PDU.RTA-SDU Retry:= M_Retry+1 A_Data_req	WACK

#	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(CREP,ERRCLS,ERRCODE)	OPEN
12	OPEN	A_Data_cnf => ignore	OPEN
13	OPEN	A_Timer_event.ind(CREP) => ignore	OPEN
14	WACK	APMS_Activate.req(CREP,DA,SA, FrameID,VLANPrio, VLANID, RTATimeoutFactor, MRetry, Transport, DstIP, SrcIP) => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_Activate.cnf (-)(CREP,ERRCLS,ERRCODE)	WACK
15	WACK	APMS_Close.req(CREP,ErrCode) /VLANPrio == Low => PDU.Version:=1 PDU.PDUType:=ERR PDU.AddFlags:=0 PDU.Window:=1 PDU.SendSeqNum:=Seq_Count PDU.AckSeqNum:=Seq_CountO of APMR PDU.PNIOStatus.ErrorCode = 0xCF PDU.PNIOStatus.ErrorDecode = 0x81 PDU.PNIOStatus.ErrorCode1 = 0xFD PDU.PNIOStatus.ErrorCode2 = ErrCode A_Data_req A_Timer_remove.req(CREP) APMS_Close.cnf (+)(CREP)	CLOSED
16	WACK	APMS_Close.req(CREP) /VLANPrio == High => A_Timer_remove.req(CREP) APMS_Close.cnf (+)(CREP)	CLOSED
17	WACK	APMS_A_Data.req(CREP, Data) => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_A_Data.cnf(-)(CREP,ERRCLS,ERRCODE)	WACK
18	WACK	A_Data_ind /(PDU.PDUType == DATA PDU.PDUType == ACK) && PDU.AckSeqNum == Seq_Count => Seq_CountO=Seq_Count Seq_Count:= (Seq_Count+1) & 0x7fff APMS_A_Data.cnf(+)(CREP)	OPEN
19	WACK	A_Data_ind /(PDU.PDUType == DATA PDU.PDUType == ACK) && PDU.AckSeqNum == Seq_CountO => ignore	WACK

#	Current State	Event /Condition =>Action	Next State
20	WACK	A_Data_ind /(PDU.PDUType == DATA PDU.PDUType == ACK) && PDU.AckSeqNum != Seq_Count && PDU.AckSeqNum != Seq_CountO => ignore	WACK
21	WACK	A_Data_ind /PDU.Type == ERR => APMS_Error.ind(CREP,ERRCLS,ERRCODE)	WACK
22	WACK	A_Data_ind /PDU.PDUType == NAK => ignore	WACK
23	WACK	A_Data_cnf /LMPM_status == OK => ignore	WACK
24	WACK	A_Data_cnf /LMPM_status != OK => ERRCLS := APMS ERRCODE := LMPM APMS_Error.ind(CREP,ERRCLS,ERRCODE)	WACK
25	WACK	A_Timer_event.ind(CREP) / Retry != 0 => PDU.Version:=1 PDU.PDUType:=DATA PDU.AddFlags.TACK:=from Stored PDU.AddFlags PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Seq_Count PDU.AckSeqNum:=Seq_CountO of APMR PDU.RTA-SDU:= from Stored PDU.RTA-SDU Retry:= Retry-1 A_Data_req	WACK
26	WACK	A_Timer_event.ind(CREP) /Retry = 0 => ERRCLS := APMS ERRCODE := TIMEOUT APMS_Error.ind(CREP,ERRCLS,ERRCODE)	OPEN

4.7.3.1.4 Functions

Table 163 contains the functions or macros used by the APMS and APMR, their arguments and their descriptions.

Table 163 – Functions used by the APMS and APMR

Name	Function
A_Data_ind	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	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	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)
A_Timer_add	Add a cyclic timer
A_Timer_event	Signals a cyclic timer event
A_Timer_remove	Removes a cyclic timer

4.7.3.1.4.1 A_Data_ind, A_Data_cnf and A_Data_req

Local macros used to send and receive UDP-RTA-PDU or RTA-PDU depending on RTA or UDP transport. With A_Data_req the D_Port is to AUTO so LMPM will select the proper output port. TStamp, S_Port is not used.

4.7.3.1.4.2 A_Timer_add

This local service will be used to add a cyclic timer in order to indicate cyclic timer events (A_Timer_event.ind) with the specified TimeFactor. The timer events will occur till the timer will be removed with the “A_Timer_remove” service.

Table 164 – A_Timer_add

Parameter name	Req
Argument	M
CREP	M
TimeFactor	M

Argument

The argument shall convey the service specific parameters of the service request.

CREP

This parameter identifies the timer for further use

Attribute Type: Unsigned32

TimeFactor

This parameter specifies the timer cycle as a multiple of 100 ms

Attribute Type: Unsigned16

Time Base: 100 ms

4.7.3.1.4.3 A_Timer_event

This local service will be called if a timer cycle elapsed.

Table 165 – A_Timer_event

Parameter name	Ind
Argument	M
CREP	M

Argument

The argument shall convey the service specific parameters of the service request.

CREP

This parameter identifies the timer.

Attribute Type: Unsigned32

4.7.3.1.4.4 A_Timer_remove

This local service will be used to remove a cyclic timer. After a timer is removed, no more "A_Timer_event" occurs.

Table 166 – A_Timer_remove

Parameter name	Req
Argument	M
CREP	M

Argument

The argument shall convey the service specific parameters of the service request.

CREP

This parameter identifies the timer to be removed.

Attribute Type: Unsigned32

4.7.3.2 APMR

4.7.3.2.1 Primitive definitions

4.7.3.2.1.1 Primitives exchanged between APMR and APMR user

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

4.7.3.2.1.2 Primitives exchanged between APMR and LMPM

The service primitives including their associated parameters issued by APMR received by LMPM and vice versa are described in the Common DL Mapping ASE in the service definition.

4.7.3.2.1.3 Parameters of APMR primitives

The parameters used with the primitives exchanged between the APMR user and the APMR are described in FAL Service Definition.

4.7.3.2.2 State machine description

The CLOSED state indicates that an initialisation is needed. The Activate service sets the machine in the OPEN state waiting for LMPM_A_Data service indication. After passing a Data indication to the user the WACK state is used to wait for acknowledge from the APMR user. Receiving acknowledge will set back the machine to the OPEN state. A Close service request sets the protocol machine to the CLOSED state.

Local variables of the APMR

PDU

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 (e.g. IP-header) is not shown.

PDUType: 1 = DATA, 2 = NACK, 3 = ACK, 4 = ERR

Seq_Count (Unsigned 16)

This local variable contains the counter value which shall be used at the receipt of the next DATA PDU.

Seq_Count0 (Unsigned 16)

This local variable contains the previous counter value of the Seq_Count variable.

Retry (Unsigned16)

This local variable counts the remaining timer ticks till a transmit retry.

4.7.3.2.3 APMR state table

Table 167 contains the complete description of the APMR state machine. A LMPM_A_Data.ind primitive will be accepted.

If PDUType.Type field is DATA-RTA-PDU and the PDUType.Version field is 1 and the VarPartLen is greater then 0 and the AddFlagsWindow is 1.

If PDUType.Type field is ERR-RTA-PDU and the PDUType.Version field is 1 and the VarPartLen is 4.

Table 167 – APMR state table

#	Current State	Event /Condition =>Action	Next State
1	CLOSED	APMR_Activate.req(CREP,DA,SA,FrameID,VLANPrio, VLANID, Transport, DstIP, SrcIP) => Seq_Count:=0xffff Seq_CountO:=0xfffe Store DA,SA,FrameID,VLANPrio, VLAN,MRetry,Transport, DstIP, SrcIP APMR_Activate.cnf (+)(CREP)	OPEN
2	CLOSED	APMR_Close.req(CREP) => APMR_Close.cnf (+)(CREP)	CLOSED
3	CLOSED	APMR_ACK.req(CREP) => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_ACK.cnf (-) (CREP,ERRCLS,ERRCODE)	CLOSED
4	CLOSED	A_Data_ind => ignore	CLOSED
5	CLOSED	A_Data_cnf => ignore	CLOSED
6	OPEN	APMR_Activate.req(CREP,DA,SA,FrameID,VLANPrio, VLANID, Transport, DstIP, SrcIP) => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_Activate.cnf (-) (CREP,ERRCLS,ERRCODE)	OPEN
7	OPEN	APMR_Close.req(CREP) => APMR_Close.cnf (+)(CREP)	CLOSED
8	OPEN	APMR_ACK.req(CREP) => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_ACK.cnf (-) (CREP,ERRCLS,ERRCODE)	OPEN
9	OPEN	A_Data_ind /PDU.Type == ERR => ERRCLS := PDU.PNIOStatus.ErrorCode1 ERRCODE := PDU.PNIOStatus.ErrorCode2 APMR_Error.ind(CREP,ERRCLS,ERRCODE)	OPEN
10	OPEN	A_Data_ind /PDU.Type == DATA && PDU.AddFlags.Tack && PDU.SendSeqNum == Seq_Count => Data := PDU.RTA-SDU APMR_A_Data.ind(CREP, Data)	WACK
11	OPEN	A_Data_ind /PDU.Type == DATA && PDU.AddFlags.Tack && PDU.SendSeqNum == Seq_CountO =>PDU.Version:=1 PDU.Type:= ACK PDU.AddFlags.TACK:=0 PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Seq_CountO of APMS PDU.AckSeqNum:=Seq_CountO A_Data_req	OPEN

#	Current State	Event /Condition =>Action	Next State
12	OPEN	A_Data_ind /PDU.Type == DATA && PDU.AddFlags.Tack && PDU.SendSeqNum != Seq_Count && PDU.SendSeqNum != Seq_CountO => ERRCLS := RTA_ERR_CLS_PROTOCOL ERRCODE := RTA_ERR_CODE_SEQ PDU.Version:=1 PDU.Type:= NAK PDU.AddFlags.TACK:=0 PDU.AddFlags.Window:=1 PDU.SendSeqNum:=Seq_CountO of APMS PDU.AckSeqNum:=Seq_CountO A_Data_req	OPEN
13	OPEN	A_Data_ind /(PDU.Type == DATA && !PDU.AddFlags.Tack) => ignore	OPEN
14	OPEN	A_Data_ind /PDU.Type == ACK PDU.Type == NAK => ignore	OPEN
15	OPEN	A_Data_cnf /LMPM_status == OK => ignore	OPEN
16	OPEN	A_Data_cnf /LMPM_status != OK => ERRCLS := APMR ERRCODE := LMPM APMR_Error.ind(CREP,ERRCLS,ERRCODE)	OPEN
17	WACK	APMR_Activate.req(CREP,DA,SA,FrameID,VLANPrio, VLANID, Transport, DstIP, SrcIP) => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_Activate.cnf (-)(CREP,ERRCLS,ERRCODE)	WACK
18	WACK	APMR_Close.req(CREP) => APMR_Close.cnf (+)(CREP)	CLOSED
19	WACK	APMR_ACK.req(CREP) =>Seq_CountO :=Seq_Count Seq_Count := (Seq_Count+1) & 0x7fff PDU.Version :=1 PDU.Type := ACK PDU.AddFlags.TACK :=0 PDU.AddFlags.WindowSize :=1 PDU.SendSeqNum :=Seq_CountO of APMS PDU.AckSeqNum :=Seq_CountO A_Data_req APMR_ACK.cnf(+)(CREP)	OPEN
20	WACK	A_Data_ind /PDU.Type != ERR => ignore	WACK
21	WACK	A_Data_ind /PDU.Type == ERR => APMR_Error.ind(CREP,ERRCLS,ERRCODE)	WACK
22	WACK	A_Data_cnf /LMPM_status == OK => ignore	WACK

#	Current State	Event /Condition =>Action	Next State
23	WACK	A_Data_cnf /LMPM_status != OK => ERRCLS := APMR ERRCODE := LMPM APMR_Error.ind(CREP,ERRCLS,ERRCODE)	WACK

4.7.3.2.4 Functions

Table 163 contains the functions or macros used by the APMS and APMR, their arguments and their descriptions.

4.8 Remote procedure call

4.8.1 RPC 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 RPC APDU abstract syntax

Table 168 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 168 shall represent the content of the DLSDU in Table 4.

Table 168 – RPC APDU syntax

APDU name	APDU structure
CL-RPC-PDU	IPHeader, UDPHeader, RPCHeader, NDRDataRequest ^ NDRDataResponse ^ NDREPMAPLookupReq ^ NDREPMAPLookupRes ^ NDREPMAPLookupFreeReq ^ NDREPMAPLookupFreeRes ^ NDRAck ^ NDRQuAck ^ NDRQuit ^ NDRFack ^ NDRFault ^ NDRNoCall ^ NDRWorking ^ NDRPing ^ NDRReject

Table 169 defines structures for substitutions of elements of the APDU structures shown in Table 168.

Table 169 – 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(0), RPCPad1, RPCWindowSize, RPCMaxTsdu, RPCMaxFragSize, RPCSerialNumber, RPCSelAckLen, RPCArrayOfSelAck*)
NDRNoCall	NULL ^ NDRFack
NDRWorking	NULL
NDRPing	NULL
NDRReject	RPCNCARjectStatus
NDREMapLookupReq	RPCInquiryType, RPCObjectReference(1), RPCObjectUUID, RPCInterfaceReference(2), RPCInterfaceUUID, RPCInterfaceVersionMajor, RPCInterfaceVersionMinor, RPCVersionOption(1), RPCEntryHandleAttribute(0), RPCEntryHandleUUID, RPCMaxEntries
NDREMapLookupRes	RPCEntryHandleAttribute(0), RPCEntryHandleUUID, RPCNumberOfEntries, RPCMaxEntries, RPCEntriesOffset, RPCEntriesCount, [(RPCObjectUUID, RPCTowerReference, RPCAnnotationOffset, RPCAnnotationLength, RPCAnnotation, [RPCGap*], RPCTowerLength, RPCTowerOctetStringLength, [RPCTowerOctetString*], [RPCGap*]*], RPCEPMapStatus
RPCTowerOctetString	RPCFloorCount, [RPCFloor*]
RPCFloor	RPCLHSByteCount, [RPCProtocolIdentifierString*], RPCRHSByteCount, [RPCRelatedData*]
RPCProtocolIdentifier-String	(RPCID, RPCInterfaceUUID ^a , RPCInterfaceVersionMajor ^a) ^ (RPCID, RPCDataRepresentationUUID ^a , RPCInterfaceVersionMajor) ^ RPCProtocolIdentifier ^ RPCServerUDPPort ^ RPCHostAddress ^a Shall be coded in little endian format
RPCRelatedData	RPCInterfaceVersionMinor ^a ^ RPCPortNumber ^ RPCIPAddress ^a Shall be coded in little endian format
NDREMapLookupFree Req	RPCEntryHandleAttribute, RPCEntryHandleUUID
NDREMapLookupFree Res	RPCEntryHandleAttribute, RPCEntryHandleUUID, RPCEPMapStatus
RPCAnnotation	DeviceType, Blank, OrderID, Blank, HWRevision, Blank, SWRevisionPrefix, SWRevision, EndTerm
^a To identify IO device, IO controller, IO supervisor ^b To identify PNIO interface type ^c To identify the service type e.g. Connect, Release, Read, Write, Control	

4.8.2 RPC Transfer syntax

4.8.2.1 Coding section related to CL-RPC-PDU

4.8.2.1.1 Coding of the field RPCVersion

This field shall be coded as data type Unsigned8. The value shall be 4.

4.8.2.1.2 Coding of the field RPCPacketType

This field shall be coded as data type Unsigned8.

The values shall be encoded according to Table 170.

Table 170 – 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.

4.8.2.1.3 Coding of the field RPCFlags

This field shall be coded as data type according to 3.7.3.3.

The values shall be encoded according to Table 171.

Table 171 – RPCFlags

Bit	Meaning if set to 1
Bit 0	Implementation specific, shall be set 0
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 0

4.8.2.1.4 Coding of the field RPCFlags2

This field shall be coded as data type according to 3.7.3.3.

The values shall be encoded according to Table 172.

Table 172 – 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

4.8.2.1.5 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 3.7.3.3 and the individual bits shall have the meaning defined in Table 173.

Table 173 – 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 3.7.3.5 the RPC encoding rules are applied to the RPCHeader, NDR substitutions and NDREMap 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 174.

Table 174 – 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.

4.8.2.1.6 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.

4.8.2.1.7 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.

4.8.2.1.8 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 175.

Table 175 – RPCObjectUUID.Data4

Octet	Meaning
1	Values see Table 176
2	
3	Instance High
4	Instance Low
5	DeviceID High
6	DeviceID Low
7	VendorID High
8	VendorID Low

Defined values for PNIO items are shown in Table 176. These values shall be used in conjunction with the values from Table 177.

Table 176 – RPCObjectUUID for PNIO

Value	Description
UUID_IO_ObjectInstance_XYZ DEA00000-6C97-11D1-8271-{xxxxyyyyzzzz}	Identifies a special object instance within a physical device in case there are more than one where
Xxxx	Represents the instance or node number
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

4.8.2.1.9 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).

Defined values for PNIO items are shown in Table 177.

Table 177 – RPCInterfaceUUID for PNIO

Value	Description
UUID_IO_DeviceInterface DEA00001-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO device uniquely.
UUID_IO_ControllerInterface DEA00002-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO controller uniquely.
UUID_IO_SupervisorInterface DEA00003-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO supervisor uniquely.
UUID_IO_ParameterServerInterface DEA00004-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO parameter server uniquely.

Defined values for the RPC end point mapper are shown in Table 178.

Table 178 – 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 0x00030000.

4.8.2.1.10 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 octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.11 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 (little endian or big endian).

NOTE Idempotent functions do not need to maintain this value. Only the endpointmapper is allowed to answer with zero.

4.8.2.1.12 Coding of the field RPCInterfaceVersion

This field shall be coded as data type Unsigned32. The value shall be set to 0x00010000. The endianness is according to the value in the field "RPCDRep.IntegerEncoding".

The major version shall be set to 1 for this version (high word).

The minor version shall be set to 0 for this version (low word).

NOTE Other values may be used for compatible extensions.

4.8.2.1.13 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 (little endian or big endian).

4.8.2.1.14 Coding of the field RPCOperationNmb

This field shall be coded as data type Unsigned16. The RPCOperationNmb identifies the PNIO service supported by the PNIO interfaces

IO device,

IO controller and

IO supervisor.

The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The values for PNIO services are defined in Table 179.

Table 179 – 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 180 shall be used.

Table 180 – 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	—

4.8.2.1.15 Coding of the field RPCInterfaceHint

This field shall be coded as data type Unsigned16. The value shall be set to no hint (0xFFFF) for this version. The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.16 Coding of the field RPCActivityHint

This field shall be coded as data type Unsigned16. The value shall be set to no hint (0xFFFF) for this version. The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.17 Coding of the field RPCLengthOfBody

This field shall be coded as data type Unsigned16. The value shall be set to the number of octets of NDRData of the current frame. The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

NOTE The conveyance of NDRData may require more than one frame. In this case RPCLengthOfBody contains only the number of octets for the current frame.

4.8.2.1.18 Coding of the field RPCFragmentNmb

This field shall be coded as data type Unsigned16. The value shall be set to the number of the current fragment. The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.19 Coding of the field RPCAuthenticationProtocol

This field shall be coded as data type Unsigned8. The value shall be set to zero for no authentication.

4.8.2.1.20 Coding of the field RPCCancelVersion

This field shall be coded as data type Unsigned32. The value shall be set to zero. The octet ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.21 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 (little endian or big endian).

4.8.2.1.22 Coding of the field RPCServerIsAccepting

This field shall be coded as data type Unsigned8.

4.8.2.1.23 Coding of the field RPCVersionFack

This field shall be coded as data type Unsigned8. The value shall be set to zero.

4.8.2.1.24 Coding of the field RPCPad1

This field shall be coded as data type Unsigned8.

4.8.2.1.25 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).

4.8.2.1.26 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 (little endian or big endian).

4.8.2.1.27 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 (little endian or big endian).

4.8.2.1.28 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 (little endian or big endian).

4.8.2.1.29 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 (little endian or big endian).

4.8.2.1.30 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 (little endian or big endian).

4.8.2.1.31 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 little endian with the value according to Table 181.

Table 181 – RPCDataRepresentationUUID – defined values

Value [UUID]	Value	Description
8A885D04-1CEB-11C9-9FE8-08002B104860	Data representation	Identifies the RPC data representation uniquely.

4.8.2.2 Coding section related to NDREPMAPDU

4.8.2.2.1 Coding of the field DeviceType

This field shall be coded as data type VisibleString[25]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 25 octets.

NOTE See also the field DeviceVendorValue.

4.8.2.2.2 Coding of the field OrderID

This field shall be coded as data type VisibleString[20]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 20 octets.

4.8.2.2.3 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".

4.8.2.2.4 Coding of the field SWRevisionPrefix

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

4.8.2.2.5 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".

NOTE As an example SWRevision could be "<Blank><Blank>1<Blank><Blank>0<Blank><Blank>0". After each group of three octets a "." should be inserted for visualization.

4.8.2.2.6 Coding of the field Blank

This field shall be coded as data type OctetString with 1 octet. The value shall be set to 0x20.

4.8.2.2.7 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 (little endian or big endian).

The coding shall be according to Table 182.

Table 182 – 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	—

4.8.2.2.8 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 (little endian or big endian).

The value shall be set to 1.

4.8.2.2.9 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.

4.8.2.2.10 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).

4.8.2.2.11 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).

4.8.2.2.12 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 (little endian or big endian).

The value shall be set to 1.

4.8.2.2.13 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 (little endian or big endian).

The value shall be set to 0.

4.8.2.2.14 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 (little endian or big endian).

4.8.2.2.15 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 (little endian or big endian).

The value shall be set at least to 1.

4.8.2.2.16 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 (little endian or big endian).

4.8.2.2.17 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 (little endian or big endian).

The value shall be set to 0.

4.8.2.2.18 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 (little endian or big endian).

4.8.2.2.19 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 (little endian or big endian).

The value shall be set to 3.

4.8.2.2.20 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 (little endian or big endian).

The value shall be set to 0.

4.8.2.2.21 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 (little endian or big endian).

The value shall be set from 0 to 64.

4.8.2.2.22 Coding of the field EndTerm

This field shall be coded as data type Unsigned8. The value shall be set to 0.

4.8.2.2.23 Coding of the field RPCGap

This field shall be coded as data type Unsigned8.

4.8.2.2.24 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 (little endian or big endian).

4.8.2.2.25 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 (little endian or big endian).

4.8.2.2.26 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 (little endian or big endian).

The coding shall be according to Table 183.

Table 183 – RPCEPMapStatus

Value (hexadecimal)	Meaning
0x00000000	RPC okay
0x16C9A0D6	Endpoint not registered
others	Reserved

4.8.2.2.27 Coding of the field RPCFloorCount

This field shall be coded as data type Unsigned16. The value shall be set to 5. The encoding shall be independently of the field RPCDRep always little endian.

4.8.2.2.28 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.

4.8.2.2.29 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.

4.8.2.2.30 Coding of the field RPCID

This field shall be coded as data type Unsigned8. The value shall be set to 0x0D.

4.8.2.2.31 Coding of the field RPCProtocolIdentifier

This field shall be coded as data type Unsigned8. The value shall be set to 0x0A.

4.8.2.2.32 Coding of the field RPCServerUDPPort

This field shall be coded as data type Unsigned8. The value shall be set to 0x08.

4.8.2.2.33 Coding of the field RPCHostAddress

This field shall be coded as data type Unsigned8. The value shall be set to 0x09.

4.8.2.2.34 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.

4.8.2.2.35 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.

4.8.2.3 Coding section related to NDRData

4.8.2.3.1 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 (little endian or big endian).

The value shall be set to the maximum buffer size available for the response.

4.8.2.3.2 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 (little endian or big endian).

The value shall be set to the number of octets within the PROFINETIOServiceReqPDU or PROFINETIOServiceResPDU.

4.8.2.3.3 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 (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.

4.8.2.3.4 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 (little endian or big endian).

The value shall be set to zero.

4.8.2.3.5 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 (little endian or big endian).

The value shall be set to the same value as in the field ArgsLength.

4.8.2.3.6 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 (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 6.2.4.68. The PNIOStatus shall be calculated according the following equation.

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

4.8.2.4 Coding section related to RPC (NCA Codes)

4.8.2.4.1 Coding of the field RPCNCAFaultStatus

The RPC specific NCA error codes shall be coded as Unsigned32 with the values according to Table 184.

Table 184 – Values of NCAFaultStatus

Value (hexadecimal)	Definition
0x1C000001	NCA_s_fault_int_div_by_zero
0x1C000002	NCA_s_fault_addr_error
0x1C000003	NCA_s_fault_fp_div_zero
0x1C000004	NCA_s_fault_fp_underflow
0x1C000005	NCA_s_fault_fp_overflow
0x1C000006	NCA_s_fault_invalid_tag
0x1C000007	NCA_s_fault_invalid_bound
0x1C000008	NCA_s_rpc_version_mismatch
0x1C000009	NCA_s_unspec_reject
0x1C00000A	NCA_s_bad_actid
0x1C00000B	NCA_s_who_are_you_failed
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

4.8.2.4.2 Coding of the field RPCNCARjectStatus

The RPC specific NCA error codes shall be coded as Unsigned32 with the values according to Table 185.

Table 185 – Values of NCARjectStatus

Value (hexadecimal)	Definition
0x1C000008	NCA_rpc_version_mismatch
0x1C000009	NCA_unspec_reject
0x1C00000A	NCA_s_bad_actid
0x1C00000B	NCA_who_are_you_failed
0x1C00000C	NCA_manager_not_entered
0x1C010002	NCA_op_rng_error
0x1C010003	NCA_unk_if
0x1C010006	NCA_wrong_boot_time
0x1C010009	NCA_s_you_crashed
0x1C01000B	NCA_proto_error
0x1C010013	NCA_out_args_too_big
0x1C010014	NCA_server_too_busy
0x1C010017	NCA_unsupported_type
0x1C00001D	NCA_unsupported_authn_level
0x1C00001F	NCA_invalid_checksum
0x1C000020	NCA_invalid_crc

4.8.3 Application Relationship Protocol Machines (ARPMs)

4.8.3.1 RPCPM

4.8.3.1.1 Primitive definitions

4.8.3.1.1.1 Primitives exchanged between RPCPM and RPCPM user

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

4.8.3.1.1.2 Primitives exchanged between RPCPM and LMPM

The service primitives including their associated parameters issued by RPCPM received by LMPM and vice versa are described in the RPC ASE in the service definition.

4.8.3.1.1.3 Parameters of RPCPM primitives

The parameters used with the primitives exchanged between the RPCPM user and the RPCPM are described in FAL Service Definition.

4.8.3.1.2 State machine description

4.8.3.1.3 RPC state table

4.8.3.1.4 Functions

4.8.3.2 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 sended 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 two seconds 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 one second 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.

4.9 Link layer discovery

4.9.1 FAL common syntax description

4.9.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.9.1.2 LLDP APDU abstract syntax

Table 186 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 186 shall represent the content of the DLSDU in Table 4.

Table 186 – 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_CHASSIS_MAC, LLDP8023MACPHY ^d , LLDPManagement, LLDP_PNIO_PTCPSTATUS ^c , [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 PTCIP 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.	

Table 187 defines structures for substitutions of elements of the APDU structures shown in Table 186.

Table 187 – LLDP substitutions

Substitution name	Structure
LLDPChassis	LLDPChassisStationName ^ LLDPChassisMacAddress ^a
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 ^c
LLDPTTL	LLDP_TLVHeader ^b , LLDP_TimeToLive(20) ^b
LLDP_PNIO_Header	LLDP_TLVHeader ^b , LLDP_OUI(00-0E-CF)
LLDP_PNIO_DELAY	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x01), PTCP_PortRxDelayLocal, PTCP_PortRxDelayRemote PTCP_PortTxDelayLocal, PTCP_PortTxDelayRemote, CableDelayLocal
LLDP_PNIO_PORTSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x02), RTClass2_PortStatus, RTClass3_PortStatus
LLDP_PNIO_ALIAS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x03), LLDP_PortID, LLDP_POINT, LLDP_ChassisID
LLDP_PNIO_MRPPORTSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x04), MRP_DomainUUID, MRRT_PortStatus
LLDP_PNIO_CHASSIS_MAC	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x05), (CMResponderMacAdd ^ CMInitiatorMacAdd) ^d
LLDP_PNIO_PTCPSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x06), PTCP_MasterSourceAddress ^e , PTCP_SubdomainUUID ^f , IRDataUUID ^g , LLDP_LengthOfPeriod ^g , LLDP_RedOrangePeriodBegin ^g , LLDP_OrangePeriodBegin ^g , LLDP_GreenPeriodBegin ^g
LLDPEnd	LLDP_TLVHeader ^b (0)
LLDP8021	LLDP_TLVHeader ^b , LLDP_OUI(00-80-C2) ^h , LLDP_8021_SubType ^h , Data ^h
LLDP8023	LLDP_TLVHeader ^b , LLDP_OUI(00-12-0F) ⁱ , LLDP_8023_SubType ⁱ , Data ⁱ
LLDP8023MACPHY	LLDP_TLVHeader ^b , LLDP_OUI(00-12-0F) ⁱ , LLDP_8023_SubType(1) ⁱ , LLDP_8023_AUTONEG ⁱ , LLDP_8023_PMDCAP ⁱ , LLDP_8023_OPMAU ⁱ
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-2005.
^c The length of the field LLDP_PortID shall be 8 or 14 octets
^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-2005 Annex F
ⁱ Shall be set according to IEEE 802.1AB-2005 Annex G
^j Shall be set according to IEEE 802.1AB-2005, 9.5.9. It is recommended to set the object identifier according to 4.15.2.

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.

4.9.2 LLDP transfer syntax

4.9.2.1 General

As an extension to IEEE 802.1AB-2005, 10.5.3.1 this standard defines that value of the field txDelayWhile shall be zero. In this case, every node transmits LLDP PDUs on data change.

4.9.2.2 Coding of the field LLDP_ChassisID

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 LLDP_ChassisID is not terminated by zero.

4.9.2.3 Coding of the field LLDP_PortID

This field shall be coded as data type OctetString with 8 or 14 octets.

This field contains the name of the local port which shall be used as Port ID subtype locally assigned compliant with IEEE 802.1AB. The value shall be “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 RFC 3490 shall be applied.

NOTE The field LLDP_PortID is not terminated by zero.

4.9.2.4 Coding of the field LLDP_PNIO_SubType

This field shall be coded as data type Unsigned8 and shall be set according to Table 188.

Table 188 – LLDP_PNIO_SubType

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Measured delay values
0x02	Port Status
0x03	Alias
0x04	MRP Port Status
0x05	Interface MAC address
0x06	PTCP Status
0x07 – 0xFF	Reserved

4.9.2.5 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 189.

Table 189 – PTCP_PortRxDelayLocal

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

4.9.2.6 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 190.

Table 190 – PTCP_PortRxDelayRemote

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

4.9.2.7 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 191.

Table 191 – PTCP_PortTxDelayLocal

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

4.9.2.8 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 192.

Table 192 – PTCP_PortTxDelayRemote

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

4.9.2.9 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 193.

Table 193 – CableDelayLocal

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

4.9.2.10 Coding of the field RTClass2_PortStatus

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: RTClass2_PortStatus.State

This field shall be set according to the Table 194.

Table 194 – RTClass2_PortStatus.State

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	—

Bit 2 – 15: RTClass2_PortStatus.reserved

This field shall be set according to 3.7.3.2.

4.9.2.11 Coding of the field RTClass3_PortStatus

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 2: RTClass3_PortStatus.State

This field shall be set according to the Table 195.

Table 195 – RTClass3_PortStatus.State

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used or configured if LLDP_RedOrangePeriodBegin.Valid is equal one. RED Phase is deactivated.
0x01	Reserved	—
0x02	RTCLASS3_UP	RED Phase activated for transmission of RTClass3 Frames. Expected next state: RTCLASS3_RUN
0x03	Reserved	—
0x04	RTCLASS3_RUN	RED Phase activated for transmission and reception of RTClass3 Frames
0x05 – 0x07	Reserved	—

Bit 3 – 15: RTClass3_PortStatus.reserved

This field shall be set according to 3.7.3.2.

4.9.2.12 Coding of the field MRRT_PortStatus

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: MRRT_PortStatus.State

This field shall be set according to the Table 196.

Table 196 – MRRT_PortStatus.State

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used / not configured
0x01	MRRT_CONFIGURED	Configured
0x02	MRRT_UP	Activated and running
0x03	Reserved	—

Bit 2 – 15: MRRT_PortStatus.reserved

This field shall be set according to 3.7.3.2.

4.9.2.13 Coding of the field IRDataUUID

This field shall be coded as data type UUID.

4.9.2.14 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 3.7.3.5 and to Figure 13. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_RedOrangePeriodBegin.Offset

This field shall be set according to the Table 197.

Table 197 – 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 optional field shall be set according to the Table 198.

Table 198 – 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.

4.9.2.15 Coding of the field LLDP_OrangePeriodBegin

As defined for RT_CLASS_2 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 3.7.3.5 and to Figure 13. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_OrangePeriodBegin.Offset

This field shall be set according to the Table 199.

Table 199 – 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 optional field shall be set according to the Table 200.

Table 200 – LLDP_OrangePeriodBegin.Valid

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.

4.9.2.16 Coding of the field LLDP_GreenPeriodBegin

As defined for RT_CLASS_1, 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 3.7.3.5 and to Figure 13. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_GreenPeriodBegin.Offset

This field shall be set according to the Table 201.

Table 201 – 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 the Table 202.

Table 202 – 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.

4.9.2.17 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 3.7.3.5 and to Figure 13. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_LengthOfPeriod.Length

This field shall be set according to the Table 203.

Table 203 – LLDP_LengthOfPeriod.Length

Value (hexadecimal)	Meaning	Usage
0x00007A12 – 0x003D0900	Duration of a cycle in nanoseconds	The value shall be a multiply of 31 250 ns. See 6.2.4.62
0x003D0900 – 0x7FFFFFFF	Reserved	—

Bit 31: LLDP_LengthOfPeriod.Valid

This optional field shall be set according to the Table 204.

Table 204 – 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.

4.9.2.18 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.

4.9.2.19 Coding of the field LLDP_TimeToLive

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.20 Coding of the field LLDP_TLVHeader

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.21 Coding of the field LLDP_ChassisIDSubType

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.22 Coding of the field LLDP_PortIDSubType

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.23 Coding of the field LLDPOption

This field shall be coded according to IEEE 802.1AB-2005, 9.4.

4.9.2.24 Coding of the field LLDP_OUI

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.25 Coding of the field LLDP_8021_SubType

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.26 Coding section related to LLDP_Management

4.9.2.26.1 Coding of the field LLDP_ManagementData

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.27 Coding section related to LLDP_8023

4.9.2.27.1 Coding of the field LLDP_8023_SubType

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.27.2 Coding of the field LLDP_8023_AUTONEG

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.27.3 Coding of the field LLDP_8023_PMDCAP

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.27.4 Coding of the field LLDP_8023_OPM AU

This field shall be coded according to IEEE 802.1AB-2005.

4.10 MAC bridges

4.10.1 Overview

The concepts according to IEEE 802.1D standard shall be applied. This part of the specification defines the utilization of the IEEE 802.1D standard. It includes extensions for RT_CLASS_3 phase regarding forwarding of frames. The routing mechanisms of IEEE 802.1D 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.

Apart from IEEE 802.1D behaviour, this protocol specification defines the following protocol machines to provide special forwarding actions:

RT_CLASS_3 Forwarding Protocol Machine (RED_RELAY)

Time Synchronization Forwarding Protocol Machine (SYNC_RELAY)

MMRT Forwarding Protocol Machine (RRT_RELAY)

4.10.2 RT_CLASS_3 port state machine (RTC3PSM)

4.10.2.1 Primitive definitions

4.10.2.1.1 Primitives exchanged between LLDP and RTC3PSM

Table 205 shows the primitives issued by the LLDP user to the RTC3PSM.

Table 205 – Primitives issued by LLDP to RTC3PSM

Primitive Name	Associated Parameters
Remote Systems Data Change.ind	Local Port ID, List of Neighbours

4.10.2.1.2 Primitives exchanged between IEEE 802.3 and RTC3PSM

Table 206 shows the primitives issued by the IEEE 802.3 to the RTC3PSM.

Table 206 – Primitives issued by IEEE 802.3 to RTC3PSM

Primitive Name	Associated Parameters
MAU Type Change.ind	Local Port ID Local MAU Type Local Link Status

4.10.2.1.3 Primitives exchanged between PTCP and RTC3PSM

Table 207 shows the primitives issued by the PTCP to the RTC3PSM.

Table 207 – Primitives issued by PTCP to RTC3PSM

Primitive Name	Associated Parameters
Sync State Change.ind	Sync State

4.10.2.1.4 Primitives exchanged between IEEE 802.1D and RTC3PSM

Table 208 shows the primitives issued by the IEEE 802.1D to the RTC3PSM.

Table 208 – Primitives issued by IEEE 802.1D to RTC3PSM

Primitive Name	Associated Parameters
Port State Change.ind	Local PortState

Table 209 shows the primitives issued by the RTC3PSM to IEEE 802.1D.

Table 209 – Primitives issued by RTC3PSM to IEEE 802.1D

Primitive Name	Associated Parameters
Set Port State.req	Local RTC3 PortState

4.10.2.2 RTC3PSM state table

The RTC3PSM state table is shown in Table 210.

Table 210 – RTC3PSM state table

#	Current State	Event /Condition =>Action	Next State
1	INIT	=> REMOTEPORTSTATE := OFF RT_CLASS_3_FAULT := FALSE	OFF
2	INIT	OTHERS ignore => ignore	INIT
3	OFF	Something changed() /CHECK_LOCAL_OK == FALSE CHECK_NEIGHBOR_OK == FALSE CHECK_SYNC_OK == FALSE => ignore	OFF
4	OFF	Something changed() /REMOTEPORTSTATE == OFF && OPTIMIZED_MODE == OFF && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => SetPortState.req(Local RT_CLASS_3 PortState == RTCLASS3_UP)	RTCLASS3_UP
5	OFF	Something changed() /REMOTEPORTSTATE == OFF && OPTIMIZED_MODE == ON && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => SetPortState.req(Local RT_CLASS_3 PortState == RTCLASS3_RUN)	RTCLASS3_RUN
6	OFF	Something changed() /REMOTEPORTSTATE == RTCLASS3_UP && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => SetPortState.req(Local RT_CLASS_3 PortState == RTCLASS3_RUN)	RTCLASS3_RUN
7	OFF	Something changed() /REMOTEPORTSTATE == RTCLASS3_RUN && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => SetPortState.req(Local RT_CLASS_3 PortState == RTCLASS3_RUN)	RTCLASS3_RUN
8	OFF	OTHERS ignore =>	OFF

#	Current State	Event /Condition =>Action	Next State
		ignore	
9	RTCLASS3_UP	Something changed() /CHECK_LOCAL_OK == FALSE CHECK_NEIGHBOR_OK == FALSE CHECK_SYNC_OK == FALSE => SetPortState.req(Local RT_CLASS_3 PortState == OFF)	OFF
10	RTCLASS3_UP	Something changed() /REMOTEPORTSTATE == OFF && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => ignore	RTCLASS3_UP
11	RTCLASS3_UP	Something changed() /REMOTEPORTSTATE == RTCLASS3_UP && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => SetPortState.req(Local RT_CLASS_3 PortState == RTCLASS3_RUN)	RTCLASS3_RUN
12	RTCLASS3_UP	Something changed() /REMOTEPORTSTATE == RTCLASS3_RUN && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => SetPortState.req(Local RT_CLASS_3 PortState == RTCLASS3_RUN)	RTCLASS3_RUN
13	RTCLASS3_UP	OTHERS ignore ==>ignore	RTCLASS3_UP
14	RTCLASS3_RUN	Something changed() /CHECK_LOCAL_OK == FALSE CHECK_NEIGHBOR_OK == FALSE CHECK_SYNC_OK == FALSE => SetPortState.req(Local RT_CLASS_3 PortState == OFF)	OFF
15	RTCLASS3_RUN	Something changed() /REMOTEPORTSTATE == OFF && OPTIMIZED_MODE == OFF && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => SetPortState.req(Local RT_CLASS_3 PortState == RTCLASS3_UP)	RTCLASS3_UP
16	RTCLASS3_RUN	Something changed() /REMOTEPORTSTATE == OFF && OPTIMIZED_MODE == ON &&	RTCLASS3_RUN

#	Current State	Event /Condition =>Action	Next State
		CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => ignore	
17	RTCLASS3_RUN	Something changed() /REMOTEPORTSTATE == RTCLASS3_UP && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => ignore	RTCLASS3_RUN
18	RTCLASS3_RUN	Something changed() /REMOTEPORTSTATE == RTCLASS3_RUN && CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => ignore	RTCLASS3_RUN
19	RTCLASS3_RUN	OTHERS ignore => ignore	RTCLASS3_RUN
20	ANY	Remote Systems Data Change(Local Port ID, List of Neighbours) /Does the remote RT_CLASS_3_Port_State exist and contain a value out of the following list {OFF, RTCLASS3_UP, RTCLASS3_RUN} => REMOTEPORTSTATE := RT_CLASS_3_Port_State Something changed()	ANY
21	ANY	Remote Systems Data Change(Local Port ID, List of Neighbours) /(// physical EXPECTED_REMOTE(LLDP_ChassisID) != REMOTE(LLDP_ChassisID) OR EXPECTED_REMOTE(LLDP_PortID) != REMOTE(LLDP_PortID) OR if activated (EXPECTED_REMOTE(MAUType) != REMOTE(LLDP_MAUType)) OR if activated (EXPECTED_REMOTE(CableDelay) < REMOTE(LLDP_CableDelay)) OR // logical EXPECTED_REMOTE(IRDataUUID) != REMOTE(LLDP_IRDataUUID) OR EXPECTED_REMOTE(PTCP_SuddomainUUID) != REMOTE(LLDP_PTCP_SuddomainUUID)) => CHECK_NEIGHBOR_OK := FALSE Something changed()	ANY
22	ANY	Remote Systems Data Change(Local Port ID, List of Neighbours) /(// physical EXPECTED_REMOTE(LLDP_ChassisID) !=	ANY

#	Current State	Event /Condition =>Action	Next State
		REMOTE(LLDP_ChassisID) OR EXPECTED_REMOTE(LLDP_PortID) != REMOTE(LLDP_PortID) OR if activated (EXPECTED_REMOTE(MAUType) != REMOTE(LLDP_MAUType)) OR if activated (EXPECTED_REMOTE(CableDelay) < REMOTE(LLDP_CableDelay)) OR // logical EXPECTED_REMOTE(IRDataUUID) != REMOTE(LLDP_IRDataUUID)) => CHECK_NEIGHBOR_OK := TRUE Something changed()	
23	ANY	MAU Type Change.ind (Local Port ID, Local MAU Type, Local Link Status) /(// physical LOCAL(MAUType) != VALID OR LOCAL(PortState) == DISCARDING OR LOCAL(LinkStatus) != UP OR // logical LOCAL(GET_DOMAIN_BOUNDARY_VALUE(DomainBoundary)) == TRUE) => CHECK_LOCAL_OK := FALSE Something changed()	ANY
24	ANY	MAU Type Change.ind (Local Port ID, Local MAU Type, Local Link Status) /(// physical LOCAL(MAUType) != VALID OR LOCAL(PortState) == DISCARDING OR LOCAL(LinkStatus) != UP OR // logical LOCAL(GET_DOMAIN_BOUNDARY_VALUE(DomainBoundary)) == TRUE) => CHECK_LOCAL_OK := TRUE Something changed()	ANY
25	ANY	Sync State Change.ind (Sync State) /(// physical LOCAL(SYNC) == FALSE OR // logical if activated (DIFFERENCE_FAULT(LOCAL(CableDelay), REMOTE(CableDelay))) OR EXPECTED_REMOTE(PTCP_SuddomainUUID) != REMOTE(LLDP_PTCP_SuddomainUUID)) => CHECK_SYNC_OK := FALSE Something changed()	ANY
26	ANY	Sync State Change.ind (Sync State) /(// physical	ANY

#	Current State	Event /Condition =>Action	Next State
		LOCAL(SYNC) == FALSE OR // logical if activated (DIFFERENCE_FAULT(LOCAL(CableDelay), REMOTE(CableDelay))) OR EXPECTED_REMOTE(PTCP_SuddomainUUID) != REMOTE(LLDP_PTCP_SuddomainUUID)) => CHECK_SYNC_OK := TRUE Something changed()	
27	ANY	Port State Change.ind (Local PortState) /(// physical LOCAL(MAUType) != VALID OR LOCAL(PortState) == DISCARDING OR LOCAL(LinkStatus) != UP OR // logical LOCAL(GET_DOMAIN_BOUNDARY_VALUE(DomainBoundary)) == TRUE) => CHECK_LOCAL_OK := FALSE Something changed()	ANY
28	ANY	Port State Change.ind (Local PortState) /(// physical LOCAL(MAUType) != VALID OR LOCAL(PortState) == DISCARDING OR LOCAL(LinkStatus) != UP OR // logical LOCAL(GET_DOMAIN_BOUNDARY_VALUE(DomainBoundary)) == TRUE) => CHECK_LOCAL_OK := TRUE Something changed()	ANY

4.10.2.3 Functions

All functions of the RTC3PSM are summarized in Table 211.

Table 211 – RTC3PSM function table

Function name	Operations
LOCAL()	This function delivers the local information.
REMOTE()	This function delivers the remote information stored locally.
EXPECTED_REMOTE()	This function delivers the local stored information which is expected to be delivered from remote.
DIFFERENCE_FAULT	This function checks the difference between the local and remote measured CableDelay.
Something changed()	This function is used as a trigger.

4.10.2.4 Generating of events

The RTC3PSM shows the state changes required to establish a RT_CLASS_3 peer to peer communication. Additional Figure 43 shows the diagnostic events, generated according to the state changes of the RTC3PSM. The Table 212 defines the coding of the associated events.

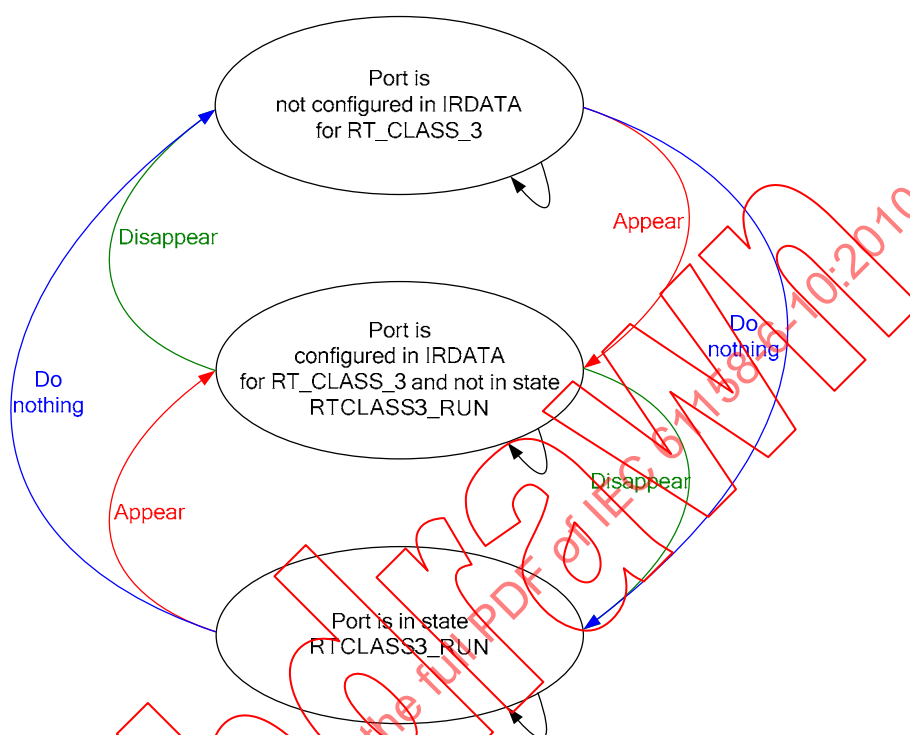


Figure 43 – State transition diagram for generating events

Table 212 – 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 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

4.10.3 RT_CLASS_2 port state machine (RTC2PSM)

4.10.3.1 Primitive definitions

4.10.3.1.1 Primitives exchanged between LLDP and RTC2PSM

Table 213 shows the primitives issued by the LLDP user to the RTC2PSM.

Table 213 – Primitives issued by LLDP to RTC2PSM

Primitive Name	Associated Parameters
Remote Systems Data Change.ind	Local Port ID, List of Neighbours

4.10.3.1.2 Primitives exchanged between IEEE 802.3 and RTC2PSM

Table 214 shows the primitives issued by the IEEE 802.3 to the RTC2PSM.

Table 214 – Primitives issued by IEEE 802.3 to RTC2PSM

Primitive Name	Associated Parameters
MAU Type Change.ind	Local Port ID Local MAU Type Local Link Status

4.10.3.1.3 Primitives exchanged between PTCP and RTC2PSM

Table 215 shows the primitives issued by the PTCP to the RTC2PSM.

Table 215 – Primitives issued by PTCP to RTC2PSM

Primitive Name	Associated Parameters
Sync State Change.ind	Sync State

4.10.3.1.4 Primitives exchanged between IEEE 802.1D and RTC2PSM

Table 216 shows the primitives issued by the IEEE 802.1D to the RTC2PSM.

Table 216 – Primitives issued by IEEE 802.1D to RTC2PSM

Primitive Name	Associated Parameters
Port State Change.ind	Local PortState

Table 217 shows the primitives issued by the RTC2PSM to IEEE 802.1D.

Table 217 – Primitives issued by RTC2PSM to IEEE 802.1D

Primitive Name	Associated Parameters
Set Port State.req	Local RTC2 PortState

4.10.3.2 RTC2PSM state table

The RTC2PSM state table is shown in Table 218.

Table 218 – RTC2PSM state table

#	Current State	Event /Condition =>Action	Next State
1	IDLE	Something changed() /CHECK_LOCAL_OK == TRUE && CHECK_NEIGHBOR_OK == TRUE && CHECK_SYNC_OK == TRUE => SetPortState.req(Local RT_CLASS_2 PortState == RTCLASS2_RUN)	IDLE
2	IDLE	Something changed() /CHECK_LOCAL_OK == FALSE CHECK_NEIGHBOR_OK == FALSE CHECK_SYNC_OK == FALSE => SetPortState.req(Local RT_CLASS_2 PortState == OFF)	IDLE
3	IDLE	OTHERS ignore => ignore	IDLE
4	IDLE	Remote Systems Data Change.ind (Local Port ID, List of Neighbours) /(// physical // logical EXPECTED_REMOTE(PTCP_SuddomainUUID) != REMOTE(LLDP_PTCP_SuddomainUUID) OR REMOTE(LLDP_OrangePeriodBegin.Valid) == FALSE) => CHECK_NEIGHBOR_OK := FALSE Something changed()	IDLE
5	IDLE	Remote Systems Data Change.ind (Local Port ID, List of Neighbours) /(// physical // logical EXPECTED_REMOTE(PTCP_SuddomainUUID) != REMOTE(LLDP_PTCP_SuddomainUUID) OR REMOTE(LLDP_OrangePeriodBegin.Valid) == FALSE) => CHECK_NEIGHBOR_OK := TRUE Something changed()	IDLE
6	IDLE	MAU Type Change.ind (Local Port ID, Local MAU Type, Local Link Status) /(// physical LOCAL(MAUType) != VALID OR LOCAL(PortState) == DISCARDING OR if MRP is not used LOCAL(PortState) == BLOCKED OR LOCAL(LinkStatus) != UP OR // logical LOCAL(LLDP_OrangePeriodBegin.Valid) == FALSE) => CHECK_LOCAL_OK := FALSE Something changed()	IDLE
7	IDLE	MAU Type Change.ind (Local Port ID, Local MAU Type, Local Link Status) /(// physical LOCAL(MAUType) != VALID OR LOCAL(PortState) == DISCARDING OR if MRP is not used LOCAL(PortState) == BLOCKED	IDLE

#	Current State	Event /Condition =>Action	Next State
		OR LOCAL(LinkStatus) != UP OR // logical LOCAL(LLDP_OrangePeriodBegin.Valid) == FALSE) => CHECK_LOCAL_OK := TRUE Something changed()	
8	IDLE	Sync State Change.ind (Sync State) /(// physical LOCAL(SYNC) == FALSE) => CHECK_SYNC_OK := FALSE Something changed()	IDLE
9	IDLE	Sync State Change.ind (Sync State) /(// physical LOCAL(SYNC) == FALSE) => CHECK_SYNC_OK := TRUE Something changed()	IDLE
10	IDLE	Port State Change.ind (Local PortState) / (// physical LOCAL(MAUType) != VALID OR LOCAL(PortState) == DISCARDING OR if MRP is not used LOCAL(PortState) == BLOCKED OR LOCAL(LinkStatus) != UP OR // logical LOCAL(LLDP_OrangePeriodBegin.Valid) == FALSE)) => CHECK_LOCAL_OK := FALSE Something changed()	IDLE
11	IDLE	Port State Change.ind (Local PortState) /(// physical LOCAL(MAUType) != VALID OR LOCAL(PortState) == DISCARDING OR if MRP is not used LOCAL(PortState) == BLOCKED OR LOCAL(LinkStatus) != UP OR // logical LOCAL(LLDP_OrangePeriodBegin.Valid) == FALSE) => CHECK_LOCAL_OK := TRUE Something changed()	IDLE

4.10.3.3 Functions

All functions of the RTC2PSM are summarized in Table 219.

Table 219 – RTC2PSM function table

Function name	Operations
LOCAL()	This function delivers the local information.
REMOTE()	This function delivers the remote information stored locally.
EXPECTED_REMOTE()	This function delivers the local stored information which is expected to be delivered from remote.
Something changed()	This function is used as a trigger.

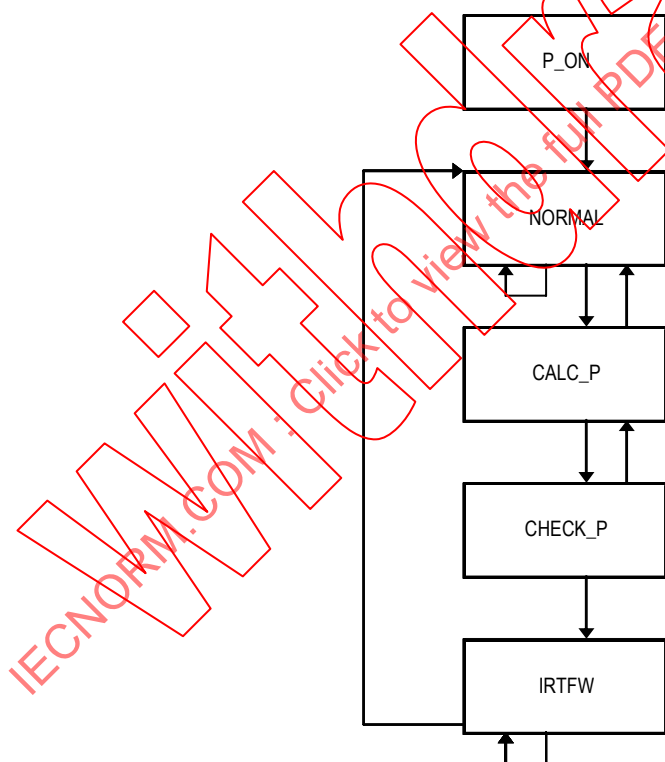
4.10.4 RT_CLASS_3 Forwarding Protocol Machine (RED RELAY)

4.10.4.1 Primitive definitions

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

4.10.4.2 RED RELAY state transition diagram

The state transition diagram of the RED RELAY is shown in Figure 44.

**Figure 44 – State transition diagram of RED RELAY**

States of the RED RELAY

P_ON
Data initialization

NORMAL
Normal operation

CALC_P

Calculate phase

CHECK_P

Check phase

IRTFW

IRT forwarding active

4.10.4.3 RED RELAY state table

The RED RELAY state table is shown in Table 220.

Withdrawn
IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 220 – RED RELAY state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	=>	NORMAL
2	NORMAL	RED_RELAY_Schedule_Req (Port, ClockTime, Period, Len) /Period == Red && Sched_List(Port) <> NIL => RedRatio := 1 L_ClockTime := ClockTime	CALC_P
3	NORMAL	RED_RELAY_Schedule_Req (Port, ClockTime, Period, Len) /Period == Red && Sched_List(Port) == NIL =>	NORMAL
4	CALC_P	/RedRatio <= MAX_REDUCTION_RATIO => i := 1 L_Phase := (L_ClockTimer MOD RedRatio) + 1	CHECK_P
5	CALC_P	/RedRatio > MAX_REDUCTION_RATIO =>	NORMAL
6	CHECK_P	/Sched_List(Port)(RedRatio)(L_Phase)(i) <> NIL => Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset)	IRTFW
7	CHECK_P	/Sched_List(Port)(RedRatio)(L_Phase)(i) == NIL => RedRatio++	CALC_P
8	IRTFW	DMUX_RED_RELAY.ind(Port, DA, SA, FrameID, DLSDU) /Sched_List(Port)(RedRatio)(L_Phase)(i) <> NIL && FrameID == Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID && DLSDU.Len == Sched_List(Port)(RedRatio)(L_Phase)(i).Len => for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_TO_RED_Q (n) get next RedRatio, i Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset)	IRTFW
9	IRTFW	DMUX_RED_RELAY.ind(Port, DA, SA, FrameID, DLSDU) /Sched_List(Port)(RedRatio)(L_Phase)(i) == NIL => DMUX_MAC_RELAY.ind(Port, DA, SA, FrameID, DLSDU)	IRTFW
10	IRTFW	DMUX_RED_RELAY.ind(Port, DA, SA, FrameID, DLSDU) /Sched_List(Port)(RedRatio)(L_Phase)(i) <> NIL && FrameID == Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID && DLSDU.Len != Sched_List(Port)(RedRatio)(L_Phase)(i).Len && !Optimized => DLSDU.Len := Sched_List(Port)(RedRatio)(L_Phase)(i).Len DLSDU.FrameID := Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID ErrCode := WRONG_LENGTH APDU_Status.TransferStatus := WRONG_LENGTH for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_TO_RED_Q (n) ----- C_SDU is undefined get next RedRatio, i Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset) RED_RELAY_Schedule_Error_Ind(CREP, ErrCode)	IRTFW
11	IRTFW	DMUX_RED_RELAY.ind(Port, DA, SA, FrameID, DLSDU) /Sched_List(Port)(RedRatio)(L_Phase)(i) <> NIL && FrameID == Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID && DLSDU.Len != Sched_List(Port)(RedRatio)(L_Phase)(i).Len && Optimized =>	IRTFW

#	Current State	Event /Condition =>Action	Next State
		RED_RELAY_Schedule_Error_Ind(CREP, ErrCode) DMUX_MAC_RELAY.ind(Port, DA, SA, FrameID, DLSDU)	
12	IRTFW	DMUX_RED_RELAY.ind(Port, DA, SA, FrameID, DLSDU) /Sched_List(Port)(RedRatio)(L_Phase)(i) <> NIL &&FrameID != Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID && !Optimized => DLSDU.Len := Sched_List(Port)(RedRatio)(L_Phase)(i).Len DLSDU.FrameID := Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID ErrCode := WRONG_FRAMEID APDU_Status.TransferStatus := WRONG_FRAMEID for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_TO_RED_Q (n) ----- C_SDU is undefined get next RedRatio, i Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset) RED_RELAY_Schedule_Error_Ind(CREP, ErrCode)	IRTFW
13	IRTFW	DMUX_RED_RELAY.ind(Port, DA, SA, FrameID, DLSDU) /Sched_List(Port)(RedRatio)(L_Phase)(i) <> NIL &&FrameID != Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID && Optimized => RED_RELAY_Schedule_Error_Ind(CREP, ErrCode) DMUX_MAC_RELAY.ind(Port, DA, SA, FrameID, DLSDU)	IRTFW
14	IRTFW	DMUX_RED_RELAY.ind(Port, DA, SA, FrameID, DLSDU) /Sched_List(Port)(RedRatio)(L_Phase)(i) == NIL && !Optimized => RED_RELAY_Schedule_Error_Ind(CREP, ErrCode)	NORMAL
15	IRTFW	DMUX_RED_RELAY.ind(Port, DA, SA, FrameID, DLSDU) /Sched_List(Port)(RedRatio)(L_Phase)(i) == NIL && Optimized => RED_RELAY_Schedule_Error_Ind(CREP, ErrCode) DMUX_MAC_RELAY.ind(Port, DA, SA, FrameID, DLSDU)	NORMAL
16	IRTFW	IRT_Timer expired /Sched_List(Port)(RedRatio)(L_Phase)(i) == NIL => ignore	NORMAL
17	IRTFW	RED_RELAY_Schedule_Req (Port, ClockTime, Period, Len) => ErrCode := RUNNING_RED_PERIOD RED_RELAY_Schedule_Error_Ind(CREP, ErrCode)	IRTFW
18	IRTFW	IRT_Timer expired Sched_List(Port)(RedRatio)(L_Phase)(i) != NIL => DLSDU.Len := Sched_List(Port)(RedRatio)(L_Phase)(i).Len DLSDU.FrameID := Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID ErrCode := IRT_ERROR APDU_Status.TransferStatus := IRT_ERROR for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_TO_RED_Q (n) ----- C_SDU is undefined get next RedRatio, i Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset)	NORMAL

4.10.4.4 Functions

All functions of the RED RELAY are summarized in Table 221.

Table 221 – RED RELAY function table

Name	Function
Sched_List	Finds the corresponding frame
RED_RELAY_Schedule_Error_Ind	An fault is detected and signaled
PUT_TO_RED_Q	This function puts the frame in the RED queue of the QueueHandler. The MUX dequeues the frame and give it to the MAC.

4.10.4.5 Macros

All macros of the RED RELAY are summarized in Table 222.

Table 222 – RED RELAY macro table

Name	Function
Optimized	If TRUE an optimized handling for "out of band" frames is possible

4.11 Virtual bridges

4.11.1 Overview

The concepts according to IEEE 802.1Q standard shall be applied. This part of the specification defines the utilization of the IEEE 802.1Q standard. It includes extensions for RT_CLASS_3 phase regarding forwarding of frames. 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.

Apart from IEEE 802.1Q standard behaviour, this protocol specification defines the following protocol machines to provide special MAC actions:

Queue handler Protocol Machine

Multiplex MAC Protocol Machine (MUX)

4.11.2 MUX

4.11.2.1 Primitive definitions

4.11.2.1.1 Primitives exchanged between MUX and MAC

Table 223 shows the service primitives including their associated parameters issued by the MMAC and received by the MAC.

Table 223 – Primitives issued by MUX to MAC

Primitive name	Source	Associated parameters	Functions
M_UNITDATA.req	MMAC	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	—

Table 224 shows the service primitives including their associated parameters issued by the MAC and received by the MUX.

Table 224 – Primitives issued by MAC to MUX

Primitive name	Source	Associated parameters	Functions
M_UNITDATA.cnf	MAC	—	—

4.11.2.1.2 Parameters of DMPM primitives

A data request primitive is invoked in order to generate a M_UNITDATA.request primitive, as defined in the Internal Sublayer Service, IEEE 802.3.

The frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority and frame_check_sequence parameters are as defined for the M_UNITDATA.req primitive of the Internal Sublayer service.

The definition of the canonical_format_indicator parameter is as defined for the E-ISS data indication. The vlan_classification parameter carries the VLAN classification assigned to the frame by the Ingress rules. The rif_information parameter, if present, carries the value of any RIF information to be associated with the request. The include_tag parameter carries a Boolean value. True indicates to the service provider that the mac_service_data_unit parameter of the data request shall include a Tag Header. False indicates that a Tag Header shall not be included. The frame_type, mac_action, destination_address, source_address and frame_check_sequence parameters carry values equal to the corresponding parameters in the received data indication.

4.11.2.2 State machine description

The MUX is responsible for mapping LMPM, RED_RELAY, MAC_RELAY(IEEE 802.1D) and RRT_RELAY services to MAC.

Additional the Queue handler shown in Figure 47 is used to managed the priority queues.

The following Figure 45 shows the MUX state machine:

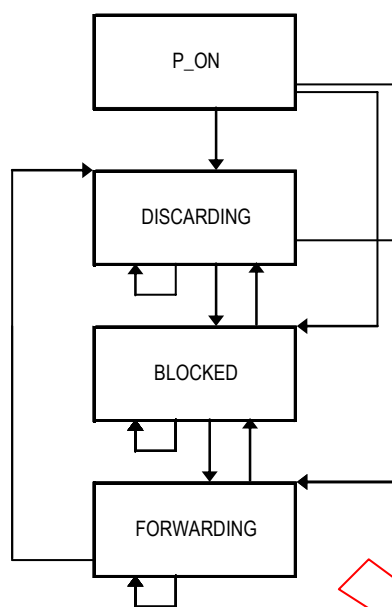


Figure 45 – State transition diagram of MUX

4.11.2.3 MUX state table

The MUX state table is shown in Table 225.

Table 225 – MUX state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	/Link_State == UP && Port_State == FORWARDING => Period := GREEN MRRT_ON := False	FORWARDING
2	P_ON	/Link_State == UP && Port_State == BLOCKED => Period := GREEN MRRT_ON := False	BLOCKED
3	P_ON	/Port_State == DISCARDING Link_State != UP => Period := GREEN MRRT_ON := False	DISCARDING
4	DISCARDING	LinkStateChange() /Link_State == UP && Port_State == BLOCKED => ignore	BLOCKED
5	DISCARDING	PortStateChange() /Link_State == UP && Port_State == FORWARDING => ignore	FORWARDING
6	DISCARDING	MUX_Schedule_req(CycleCounter, Tx_Period) => Period := GREEN	DISCARDING
7	DISCARDING	Queues_not_empty() => GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo, PRI07, RED, ORANGE, PRI06, ..., PRIOR2-1) DISCARD_FRAME()	DISCARDING
8	BLOCKED	Queues_not_empty() /Period == YELLOW && !MRRT_ON && FRAME_SIZE_FITS(NwC_Hi, NwC_Lo) => GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	BLOCKED
9	BLOCKED	Queues_not_empty() /Period == YELLOW && MRRT_ON && FRAME_SIZE_FITS(NwC_Hi, NwC_Lo, RED, ORANGE) => GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo, RED, ORANGE) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	BLOCKED
10	BLOCKED	Queues_not_empty() /Tx_Period == YELLOW && MRRT_ON && !FRAME_SIZE_FITS(NwC_Hi, NwC_Lo, RED, ORANGE) => ignore	BLOCKED
11	BLOCKED	Queues_not_empty() /Tx_Period == YELLOW && !MRRT_ON && !FRAME_SIZE_FITS(NwC_Hi, NwC_Lo) => ignore	BLOCKED
12	BLOCKED	Queues_not_empty() /Period == GREEN && !MRRT_ON =>	BLOCKED

#	Current State	Event /Condition =>Action	Next State
		GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	
13	BLOCKED	Queues_not_empty() /Period == GREEN && MRRT_ON => GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo, RED, ORANGE) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	BLOCKED
14	BLOCKED	Queues_not_empty() /Period == ORANGE && !MRRT_ON => GET_FRAME_FROM_QUEUE(NwC_Hi) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	BLOCKED
15	BLOCKED	Queues_not_empty() /Period == ORANGE && MRRT_ON => GET_FRAME_FROM_QUEUE(NwC_Hi, RED, ORANGE) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	BLOCKED
16	BLOCKED / FORWARDING	Queues_not_empty() /Period == RED => GET_FRAME_FROM_QUEUE(RED) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	BLOCKED / FORWARDING
17	FORWARDING	Queues_not_empty() /Period == YELLOW && FRAME_SIZE_FITS(NwC_Hi, NwC_Lo, PRIO7, RED, ORANGE, PRIO6, ..., PRIOR2-1) => GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo, PRIO7, RED, ORANGE, PRIO6, ..., PRIOR2-1) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	FORWARDING
18	FORWARDING	Queues_not_empty() /Period == YELLOW && !FRAME_SIZE_FITS(NwC_Hi, NwC_Lo, PRIO7, RED, ORANGE, PRIO6, ..., PRIOR2-1) => ignore	FORWARDING
19	FORWARDING	Queues_not_empty() /Period == GREEN => GET_FRAME_FROM_QUEUE(NwC_Hi, NwC_Lo, PRIO7, RED, ORANGE, PRIO6, ..., PRIOR2-1) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	FORWARDING
20	FORWARDING	Queues_not_empty() /Period == ORANGE => GET_FRAME_FROM_QUEUE(NwC_Hi, RED, ORANGE) M_UNITDATA.req (frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority,	FORWARDING

#	Current State	Event /Condition =>Action	Next State
		access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)	
21	BLOCKED / FORWARDING	MUX_Schedule_req(CycleCounter, Tx_Period) => Period := Tx_Period	BLOCKED / FORWARDING
22	BLOCKED / FORWARDING	/Period != RED && FrameEgressFilter => GET_FRAME_FROM_QUEUE() DISCARD_FRAME()	BLOCKED / FORWARDING
23	ANY	LinkStateChange() /Link_State != UP => Period := GREEN	DISCARDING
24	ANY	PortStateChange() /Port_State == Discarding => Period := GREEN	DISCARDING
25	ANY	PortStateChange() /Link_State == UP && Port_State == FORWARDING => ignore	FORWARDING
26	ANY	PortStateChange() /Link_State == UP && Port_State == BLOCKED => ignore	BLOCKED
27	ANY	PortStateChange() /MRRT_Port_State == ON => MRRT_ON := True	SAME
28	ANY	PortStateChange() /MRRT_Port_State == OFF => MRRT_ON := False	SAME
29	ANY	M_UNITDATA.cnf => ignore	SAME

4.11.2.4 Functions

All functions of the MUX are summarized in Table 226.

Table 226 – MUX function table

Name	Function
Queues_not_empty()	The QueueHandler returns TRUE if an entry exists
Queues_empty()	The QueueHandler returns TRUE if no entry exists
FRAME_SIZE_FITS(...)	Checks if the found frame fits into the remaining YELLOW period. It was only searched in the (...) queues from highest to lowest priority. The first non fitting frame stops the search and FALSE was returned.
GET_FRAME_FROM_QUEUE(...)	Delivers a frame out of (...) queues. This dequeuing always works from highest to lowest priority of the queues.

4.11.2.5 Macros

MUX defines no macros.

4.11.3 DEMUX

4.11.3.1 Primitive definitions

Table 227 shows the service primitives including their associated parameters issued by the MAC and received by the DEMUX.

Table 227 – Primitives issued by MAC to DEMUX

Primitive name	Source	Associated parameters	Functions
M_UNITDATA.ind	MAC	frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier, rif_information (optional)	—

4.11.3.1.1 Parameters of DMPM primitives

A data request primitive is invoked in order to generate a M_UNITDATA.request primitive, as defined in the Internal Sublayer Service, IEEE 802.3.

The frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority and frame_check_sequence parameters are as defined for the M_UNITDATA.req primitive of the Internal Sublayer service.

The definition of the canonical_format_indicator parameter is as defined for the E-ISS data indication. The vlan_classification parameter carries the VLAN classification assigned to the frame by the Ingress rules. The rif_information parameter, if present, carries the value of any RIF information to be associated with the request. The include_tag parameter carries a Boolean value. True indicates to the service provider that the mac_service_data_unit parameter of the data request shall include a Tag Header. False indicates that a Tag Header shall not be included. The frame_type, mac_action, destination_address, source_address and frame_check_sequence parameters carry values equal to the corresponding parameters in the received data indication.

4.11.3.2 State machine description

The DEMUX is responsible for mapping MAC services to LMPM, RED_RELAY, MAC_RELAY(IEEE 802.1D) and RRT_RELAY.

The following Figure 46 shows the DEMUX state machine:

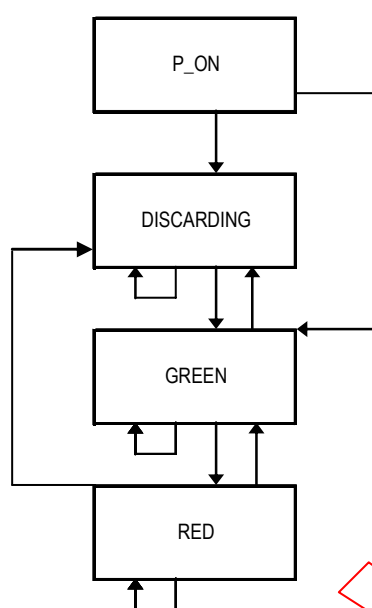


Figure 46 – State transition diagram of DEMUX

4.11.3.3 DEMUX state table

The DEMUX state table is shown in Table 228.

Table 228 – 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	PortStateChange() /Port_State != Discarding => ignore	GREEN
4	DISCARDING	M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier) => ignore	DISCARDING
5	GREEN	PortStateChange() /Port_State != Discarding => ignore	GREEN
6	GREEN	M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier) /RRT && MRRT_ON => DMUX_RRT_RELAY.ind (Port, TStamp, DA, SA, N_SDU)	GREEN
7	GREEN	M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier) /OTHER RED (RRT && !MRRT_ON) => DMUX_MAC_RELAY.ind (Port, TStamp, DA, SA, N_SDU)	GREEN
8	GREEN	M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier) /CTRL => Get_Receive_TimeStamp() DMUX_P_Data.ind (Port, TStamp, DA, SA, N_SDU)	GREEN
9	GREEN	M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier) /FrameIngressFilter \\ Checking the ingress filter is allways the first check! => ignore	GREEN
10	RED	PortStateChange() /Port_State != Discarding => ignore	RED
11	RED	M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier)/RED=>DMUX_RED_RELAY.ind (Port, TStamp, DA, SA, N_SDU)	RED

#	Current State	Event /Condition =>Action	Next State
12	RED	M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier) /RRT && MRRT_ON => DMUX_RRT_RELAY.ind (Port, TStamp, DA, SA, N_SDU)	RED
13	RED	M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier) /OTHER (RRT && !MRRT_ON) => DMUX_MAC_RELAY.ind (Port, TStamp, DA, SA, N_SDU)	RED
14	RED	M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier) /CTRL => Get_Receive_TimeStamp() DMUX_P_Data.ind (Port, TStamp, DA, SA, N_SDU)	RED
15	ANY	PortStateChange() /Port_State == Discarding => ignore	DISCARDING
16	ANY	PortStateChange() /MRRT_Port_State == ON => MRRT_ON := True	SAME
17	ANY	PortStateChange() /MRRT_Port_State == OFF => MRRT_ON := False	SAME
18	ANY	DMUX_Schedule_req (Rx_Period) /Rx_Period == GREEN => ignore	GREEN
19	ANY	DMUX_Schedule_req (Rx_Period) /Rx_Period == RED => ignore	RED

4.11.3.4 Functions

All functions of the DEMUX are summarized in Table 229.

Table 229 – DEMUX function table

Name	Function
Get_Receive_TimeStamp()	Get the receive time stamp frame the Timestamp Shim for this frame

4.11.3.5 Macros

All macros of the DEMUX are summarized in Table 230.

Table 230 – DEMUX macro table

Name	Function
RED	RT_CLASS_3 frames and PTCP synchronization frames with "RT_CLASS_3 attribute", received in RED PERIOD
CTRL	Network control/management e.g. PTCP synchronization frames without "RT_CLASS_3 attribute", LLDP, PTCP Line delay measurement, PTCP announce, media redundancy, ...
OTHER	Any frame
RRT	RT_CLASS_1, RT_CLASS_2 frames and RT_CLASS_3 frames received outside the RED PERIOD
MRRT_ON	If TRUE, the MRRT mode of this port is enabled

4.12 IP suite

4.12.1 Overview

The utilization of the RFC 768 (UDP), RFC 791 (IP), RFC 792 (ICMP), RFC 826 (ARP), RFC 1112 (IP Multicasting), RFC 2474 (IP Extensions) standards 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.

4.12.2 IP/UDP syntax description

4.12.2.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.12.2.2 IP/UDP APDU abstract syntax

Table 231 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 231 shall represent the content of the DLSDU in Table 4.

Table 231 – IP/UDP APDU syntax

APDU name	APDU structure
ICMP-PDU	IPHeader ^a , ICMPHeader, Data*
^a The value of IPHeader.IP_Protocol shall be set to 1 to indicate ICMP.	

Table 232 defines structures for substitutions of elements of the APDU structures shown in Table 231.

Table 232 – IP/UDP substitutions

Substitution name	Structure
IPHeader	IP_VersionIHL(0x45), IP_TypeOfService, 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 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.	

4.12.3 IP/UDP transfer syntax

4.12.3.1 Coding section related to IP, ICMP, and UDP

4.12.3.1.1 Coding section related to ICMP

4.12.3.1.1.1 Coding of the field ICMP_Type

The encoding of the fields shall be according to RFC 792.

4.12.3.1.1.2 Coding of the field ICMP_Code

The encoding of the fields shall be according to RFC 792.

4.12.3.1.1.3 Coding of the field ICMP_HeaderChecksum

The encoding of the fields shall be according to RFC 792.

4.12.3.1.2 Coding section related to UDP

4.12.3.1.2.1 Coding of the field UDP_DataLength

The encoding of the fields shall be according to RFC 768.

4.12.3.1.2.2 Coding of the field UDP_Checksum

The encoding of the fields shall be according to RFC 768.

4.12.3.1.2.3 Coding of the field UDP_SrcPort

The encoding of the fields shall be according to RFC 768. Defined values are shown in Table 233.

Table 233 – UDP_SrcPort

Value (hexadecimal)	Meaning	Usage
0x8892	IANA_PNIO_UDP_UNICAST_PORT	UDP-RTC-PDU and UDP-RTA-PDU
0x8893	IANA_PNIO_UDP_MULTICAST_PORT	Reserved
0x8894	IANA_PNIO_EPM_PORT	NDREPMAPLookupReq or NDREPMAPLookupFreeReq
0xC000 – 0xFFFF	IANA_FREE_PORT Is used as dynamic and/or private port	PROFINETIO→ServiceReqPDU and PROFINETIO→ServiceResPDU

4.12.3.1.2.4 Coding of the field UDP_DstPort

The encoding of the fields shall be according to RFC 768. Defined values are shown in Table 234.

Table 234 – UDP_DstPort

Value (hexadecimal)	Meaning	Usage
0x8892	IANA_PNIO_UDP_UNICAST_PORT	UDP-RTC-PDU and UDP-RTA-PDU
0x8893	IANA_PNIO_UDP_MULTICAST_PORT	Reserved
0x8894	IANA_PNIO_EPM_PORT	NDREPMAPLookupReq or NDREPMAPLookupFreeReq, also possible for each RPC call
0xC000 – 0xFFFF	IANA_FREE_PORT Is used as dynamic and/or private port	PROFINETIO→ServiceReqPDU and PROFINETIO→ServiceResPDU

4.12.3.1.3 Coding section related to IP**4.12.3.1.3.1 Coding of the field IP_DstIPAddress**

The encoding of the fields shall be according to RFC 791, RFC 1112, and RFC 2365. Defined values are shown in Table 235 and Table 236.

Table 235 – 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 236 – IP Multicast DstIPAddress according to RFC 2365

Multicast IP address	Associated Multicast MAC address	Associated FrameID	Usage
239.0.0.0 – 239.192.247.255	—	—	not used
239.192.248.0	01-00-5E-40-F8-00	0xF800	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.192.248.1 – 239.192.251.254	01-00-5E-40-F8-01 – 01-00-5E-40-FB-FE	0xF801 – 0xFBFE	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.192.251.255	01-00-5E-40-FB-FF	0xFBFF	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.193.252.0 – 239.255.255.255	—	—	not used

4.12.3.1.3.2 Coding of the field IP_VersionIHL

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.3 Coding of the field IP_TypeOfService

The encoding of the fields shall be according to RFC 791 and RFC 2474.

4.12.3.1.3.4 Coding of the field IP_TotalLength

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.5 Coding of the field IP_Identification

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.6 Coding of the field IP_Flags_FragOffset

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.7 Coding of the field IP_TTL

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.8 Coding of the field IP_Protocol

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.9 Coding of the field IP_HeaderChecksum

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.10 Coding of the field IP_SrcIPAddress

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.11 Coding of the field IP_Options

The encoding of the fields shall be according to RFC 791.

4.13 Domain name system

The utilization of the RFC 1034 (DNS) standard is defined for this specification. It includes the usage and the content syntax of domain name.

4.14 Dynamic host configuration

The utilization of the RFC 2131 (DHCP) standard is defined for this specification. It includes the usage and the content syntax of Client ID.

4.15 Simple network management

4.15.1 Overview

The utilization of the RFC 2674 (Bridges with traffic classes), RFC 2737 (MIB 2), RFC 2863 (IF MIB), RFC 3418 (SNMP), RFC 3621 (Power over Ethernet MIB), and RFC 3636 (MAU MIB) standards are defined for this specification. It includes the usage of different MIBs. Furthermore, the PNIO MIB in 4.15.3 and the LLDP EXT MIB is specified in 4.15.4.

4.15.2 Enterprise number

The value shall be according to the Table 237.

Table 237 – Enterprise number

Value (decimal)	Meaning
24 686	Enterprise number for PNIO MIB

4.15.3 PNIO MIB

NOTE This chapter is intentionally left blank.

4.15.4 LLDP EXT MIB

```

LLDP-EXT-PNO-MIB DEFINITIONS ::= BEGIN

IMPORTS
    SnmpAdminString          FROM SNMP-FRAMEWORK-MIB
    Unsigned32,
    MODULE-IDENTITY,
    OBJECT-TYPE
    MacAddress,
    DisplayString,
    TruthValue
    FROM SNMPv2-TC
    MODULE-COMPLIANCE,
    OBJECT-GROUP
    FROM SNMPv2-CONF
    lldpExtensions,
    lldpRemTimeMark,
    lldpRemLocalPortNum,
    lldpRemIndex,
    lldpPortConfigEntry,
    lldpLocPortNum
    FROM LLDP-MIB;

lldpXPnoMIB
    LAST-UPDATED "200730090000Z" -- September 30, 2007
    ORGANIZATION "PROFIBUS International (PNO)"
    CONTACT-INFO
        "
        URL: http://www.profibus.com
        email: info@profibus.com

        Postal: Haid-und-Neu-Strasse 7
                D-76131 Karlsruhe

        Tel: ++49 721 9658 - 590
        "
    DESCRIPTION
        "
        The LLDP Management Information Base extension module for
        PROFINET organizationally defined discovery information.

        Copyright (C) PROFIBUS Nutzerorganisation e.V. (2005).
        "
    REVISION "200612300000Z" -- December 30, 2006
    DESCRIPTION "initial version"
-- OUI for PNO 3791 (00-0E-CF)
::= { lldpExtensions 3791 }

```

```
-- Organizationally Defined Information Extension - PNO
--
-----
lldpXPnoObjects          OBJECT IDENTIFIER ::= { lldpXPnoMIB 1 }

-- LLDP PNO extension MIB groups
lldpXPnoConfig           OBJECT IDENTIFIER ::= { lldpXPnoObjects 1 }
lldpXPnoLocalData        OBJECT IDENTIFIER ::= { lldpXPnoObjects 2 }
lldpXPnoRemoteData       OBJECT IDENTIFIER ::= { lldpXPnoObjects 3 }

-----
-- PNO - Configuration
-----

lldpXPnoConfigTable      OBJECT-TYPE
    SYNTAX                SEQUENCE OF LldpXPnoConfigEntry
    MAX-ACCESS             not-accessible
    STATUS                 current
    DESCRIPTION            "
        A table that controls selection of LLDP
        TLVs to be transmitted on individual ports.
    "
    ::= { lldpXPnoConfig 1 }

lldpXPnoConfigEntry      OBJECT-TYPE
    SYNTAX                LldpXPnoConfigEntry
    MAX-ACCESS             not-accessible
    STATUS                 current
    DESCRIPTION            "
        LLDP configuration information that controls the
        transmission of PNO organizationally defined TLVs
        on LLDP transmission capable ports.

        This configuration object augments the
        lldpPortConfigEntry of the LLDP-MIB, therefore it
        is only present along with the port configuration
        defined by the associated lldpPortConfigEntry entry.

        Each active lldpXPnoConfigEntry must be restored
        from non-volatile storage (along with the corresponding
        lldpPortConfigEntry) after a re-initialization of the
        management system.
    "
    AUGMENTS               { lldpPortConfigEntry }
    ::= { lldpXPnoConfigTable 1 }

LldpXPnoConfigEntry ::=
    SEQUENCE {
        lldpXPnoConfigSPDTEnable TruthValue,
        lldpXPnoConfigPortStatusTxEnable TruthValue,
        lldpXPnoConfigAliasTxEnable TruthValue,
        lldpXPnoConfigMrpTxEnable TruthValue,
        lldpXPnoConfigPtcpTxEnable TruthValue
    }

lldpXPnoConfigSPDTEnable OBJECT-TYPE
    SYNTAX                TruthValue
    MAX-ACCESS             read-write
    STATUS                 current
    DESCRIPTION            "
        The lldpXPnoConfigSPDTEnable, which is defined as
        a truth value and configured by the network management,
        determines whether the PNO organizationally defined
        signal propagation delay TLV transmission is allowed on
        a given LLDP transmission capable port.

        The signal propagation delay is composed of the port
        transmission delay, the port receiving delay and the line
        delay. These values can't be transmitted independently of
        each other.

        The value of this object must be restored from non-volatile
        storage after a re-initialization of the management system.
    "
    DEFVAL                 { true }
    ::= { lldpXPnoConfigEntry 1 }

lldpXPnoConfigPortStatusTxEnable OBJECT-TYPE
    SYNTAX                TruthValue
    MAX-ACCESS             read-write
    STATUS                 current
    DESCRIPTION            "
        The lldpXPnoConfigPortStatusTxEnable, which is defined as
        a truth value and configured by the network management,
        determines whether the PNO organizationally defined
        RT port status TLV transmission is allowed on a given
        LLDP transmission capable port.

        The value of this object must be restored from non-volatile
        storage after a re-initialization of the management system.
    "
    DEFVAL                 { true }
    ::= { lldpXPnoConfigEntry 2 }

lldpXPnoConfigAliasTxEnable OBJECT-TYPE
    SYNTAX                TruthValue
    MAX-ACCESS             read-write
    STATUS                 current
    DESCRIPTION            "
        The lldpXPnoConfigAliasTxEnable, which is defined as
        a truth value and configured by the network management,
        determines whether the PNO organizationally defined
        alias TLV (chassisId) transmission is allowed on a given
        LLDP transmission capable port.
    "
```

The value of this object must be restored from non-volatile storage after a re-initialization of the management system.

```

DEFVAL
 ::= { lldpXPnoConfigEntry 3 }

lldpXPnoConfigMrpTxEnable OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-write
STATUS current
DESCRIPTION
 "
 The lldpXPnoConfigMrpTxEnable, which is defined as
 a truth value and configured by the network management,
 determines whether the PNO organizationally defined
 MRP TLV transmission is allowed on a given
 LLDP transmission capable port.

The value of this object must be restored from non-volatile
storage after a re-initialization of the management system.
 "
DEFVAL
 ::= { lldpXPnoConfigEntry 4 }

lldpXPnoConfigPtcpTxEnable OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-write
STATUS current
DESCRIPTION
 "
 The lldpXPnoConfigPtcpTxEnable, which is defined as
 a truth value and configured by the network management,
 determines whether the PNO organizationally defined
 PTCIP TLV transmission is allowed on a given
 LLDP transmission capable port.

The value of this object must be restored from non-volatile
storage after a re-initialization of the management system.
 "
DEFVAL
 ::= { lldpXPnoConfigEntry 5 }

```

-- PNO - Local System Information

```

lldpXPnoLocTable OBJECT-TYPE
SYNTAX SEQUENCE OF LldpXPnoLocEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
 "
 This table contains one row per port for PNO
 organizationally defined LLDP extension on the
 local system known to this agent.
 "
 ::= { lldpXPnoLocalData 1 }

lldpXPnoLocEntry OBJECT-TYPE
SYNTAX LldpXPnoLocEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
 "
 Additional information about a particular port
 component.

This object is indexed by the lldpLocPortNum
of the LLDP-MIB, therefore it is only present
along with the port entry defined by the
associated lldpLocPortEntry entry.

Each active lldpXPnoLocEntry must be restored
from non-volatile storage (along with the
corresponding lldpLocPortEntry) after a
re-initialization of the management system.
 "
INDEX
 ::= { lldpXPnoLocTable 1 }

LldpXPnoLocEntry ::=
 lldpXPnoLocLPDValue Unsigned32,
 lldpXPnoLocPortTxDValue Unsigned32,
 lldpXPnoLocPortRxDValue Unsigned32,
 lldpXPnoLocPortStatusRT2 INTEGER,
 lldpXPnoLocPortStatusRT3 INTEGER,
 lldpXPnoLocPortNoS DisplayString,
 lldpXPnoLocPortMrpUuId OCTET STRING,
 lldpXPnoLocPortMrpStatus INTEGER,
 lldpXPnoLocPortPtcpMaster MacAddress,
 lldpXPnoLocPortPtcpSubdomainUUID OCTET STRING,
 lldpXPnoLocPortPtcpIRDataUUID OCTET STRING,
 lldpXPnoLocPortModeRT3 INTEGER,
 lldpXPnoLocPortPeriodLength Unsigned32,
 lldpXPnoLocPortPeriodValidity TruthValue,
 lldpXPnoLocPortRedOffset Unsigned32,
 lldpXPnoLocPortRedValidity TruthValue,
 lldpXPnoLocPortOrangeOffset Unsigned32,
 lldpXPnoLocPortOrangeValidity TruthValue,
 lldpXPnoLocPortGreenOffset Unsigned32,
 lldpXPnoLocPortGreenValidity TruthValue
}

lldpXPnoLocLPDValue OBJECT-TYPE
SYNTAX Unsigned32
UNITS "ns"
MAX-ACCESS read-only
STATUS current
DESCRIPTION
 "

```

This integer value represents the line propagation delay in nanoseconds which was measured by the local system on the corresponding port.

A value of zero shall be used if the system either could not accomplish the measurement or does not support such a measurement.

```

DEFVAL
::= { lldpXPnoLocEntry 1 }

lldpXPnoLocPortTxDValue
SYNTAX          OBJECT-TYPE
                  Unsigned32
UNITS            "ns"
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION      "
                  This integer value represents the PortTxDelay
                  in nanoseconds which was measured by the local
                  system on the corresponding port.

                  A value of zero shall be used if the system either
                  could not accomplish the measurement or does not
                  support such a measurement.
                  "

DEFVAL
::= { lldpXPnoLocEntry 2 }

lldpXPnoLocPortRxDValue
SYNTAX          OBJECT-TYPE
                  Unsigned32
UNITS            "ns"
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION      "
                  This integer value represents the PortRxDelay
                  in nanoseconds which was measured by the local
                  system on the corresponding port.

                  A value of zero shall be used if the system either
                  could not accomplish the measurement or does not
                  support such a measurement.
                  "

DEFVAL
::= { lldpXPnoLocEntry 3 }

lldpXPnoLocPortStatusRT2
SYNTAX          OBJECT-TYPE
                  INTEGER {
off(0),
configured(1),
running(2)
}
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION      "
                  This value represents the status of the corresponding
                  port of the local system according to RT class 2.

                  A value of off(0) means that there isn't any RT2
                  capability available for this port. When the port is
                  configured for RT2 mode, but the mode isn't active yet
                  the value will be configured(1). If the RT2 mode is
                  configured for this port and the mode is active, the
                  value will be running(2).
                  "

::= { lldpXPnoLocEntry 4 }

lldpXPnoLocPortStatusRT3
SYNTAX          OBJECT-TYPE
                  INTEGER {
off(0),
configured(1),
up(2),
down(3),
running(4)
}
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION      "
                  This value represents the status of the corresponding
                  port of the local system according to RT class 3.

                  A value of off(0) means that there isn't any RT3
                  capability available for this port. When the port is
                  configured for RT3 mode, but the mode isn't active yet
                  the value will be configured(1).
                  When the port is ready for transmission and reception
                  of RT3 traffic, the port status will be running(4).
                  "

::= { lldpXPnoLocEntry 5 }

lldpXPnoLocPortNoS
SYNTAX          OBJECT-TYPE
                  DisplayString
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION      "
                  The local PROFINET NameofStation. If the station isn't
                  configured yet, the value of this object will be the
                  MAC address of the device as a string.
                  "

::= { lldpXPnoLocEntry 6 }

lldpXPnoLocPortMrpUuId
SYNTAX          OBJECT-TYPE
                  OCTET STRING (SIZE (16))
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION      "

```

The UUID of the MRP domain to which this port belongs to. If the port doesn't belong to a MRP domain, the value must be NIL ('0000000000000000').

```

 ::= { lldpXPnoLocEntry 7 }

lldpXPnoLocPortMrtrtStatus OBJECT-TYPE
SYNTAX INTEGER {
    off(0),
    configured(1),
    up(2)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION "
    This object reports the status of the MRRT entity of the
    corresponding port.

    A value of off(0) means that there isn't any MRRT
    capability available for this port or it is switched off.
    The value configured(1) indicates that MRRT is configured
    for the port. When MRRT is active on the port, the value
    will be up(2).
"

 ::= { lldpXPnoLocEntry 8 }

lldpXPnoLocPortPtcpMaster OBJECT-TYPE
SYNTAX MacAddress
MAX-ACCESS read-only
STATUS current
DESCRIPTION "
    The interface MAC address of the PTCP sync master device
    of the PTCP subdomain. If unknown it shall be set to zero.
"

 ::= { lldpXPnoLocEntry 9 }

lldpXPnoLocPortPtcpSubdomainUUID OBJECT-TYPE
SYNTAX OCTET STRING (SIZE (16))
MAX-ACCESS read-only
STATUS current
DESCRIPTION "
    The UUID of the PTCP subdomain to which this port belongs
    to. If the port doesn't belong to a PTCP subdomain or the subdomain
    is invalid or unknown the value of this object must be
    NIL ('0000000000000000').
"

 ::= { lldpXPnoLocEntry 10 }

lldpXPnoLocPortPtcpIRDataUUID OBJECT-TYPE
SYNTAX OCTET STRING (SIZE (16))
MAX-ACCESS read-only
STATUS current
DESCRIPTION "
    The UUID of the IR data domain to which this port belongs
    to. If the port doesn't belong to a IR data domain or the domain
    is invalid or unknown the value of this object must be
    NIL ('0000000000000000').
"

 ::= { lldpXPnoLocEntry 11 }

lldpXPnoLocPortModeRT3 OBJECT-TYPE
SYNTAX INTEGER {
    standard(1),
    optimized(0)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION "
    The mode in which the RT3 status is given. If the object is
    in standard mode all five values of RT3 status are valid, else only
    ...
"

 ::= { lldpXPnoLocEntry 12 }

lldpXPnoLocPortPeriodLength OBJECT-TYPE
SYNTAX Unsigned32 (31250..4000000)
UNITS "ns"
MAX-ACCESS read-only
STATUS current
DESCRIPTION "
    This integer value represents the duration of a cycle
    in nanoseconds on the corresponding port.
    The value shall be a multiply of 31250 nanoseconds.
"

 ::= { lldpXPnoLocEntry 13 }

lldpXPnoLocPortPeriodValidity OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-only
STATUS current
DESCRIPTION "
    The value of this object indicates whether the value of
    lldpXPnoLocPortPeriodLength of the according table entry is
    valid or not.
"

 ::= { lldpXPnoLocEntry 14 }

lldpXPnoLocPortRedOffset OBJECT-TYPE
SYNTAX Unsigned32 (0..3999999)
UNITS "ns"
MAX-ACCESS read-only
STATUS current
DESCRIPTION "
    This integer value represents the begin of the RT_CLASS_3 period

```

of the cycle of the receive direction on the corresponding port as an offset relative to the begin of the cycle in nanoseconds.

```

"
 ::= { lldpXPnoLocEntry 15 }

lldpXPnoLocPortRedValidity OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"
The value of this object indicates whether the value of
lldpXPnoLocPortRedOffset of the according table entry is
valid or not.
"

 ::= { lldpXPnoLocEntry 16 }

lldpXPnoLocPortOrangeOffset OBJECT-TYPE
SYNTAX Unsigned32 (0..3999999)
UNITS "ns"
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"
This integer value represents the begin of the RT_CLASS_2 period
of the cycle of the receive direction on the corresponding port
as an offset relative to the begin of the cycle in nanoseconds.
"

 ::= { lldpXPnoLocEntry 17 }

lldpXPnoLocPortOrangeValidity OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"
The value of this object indicates whether the value of
lldpXPnoLocPortOrangeOffset of the according table entry is
valid or not.
"

 ::= { lldpXPnoLocEntry 18 }

lldpXPnoLocPortGreenOffset OBJECT-TYPE
SYNTAX Unsigned32 (0..3999999)
UNITS "ns"
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"
This integer value represents the begin of the unrestricted period
of the cycle of the receive direction on the corresponding port
as an offset relative to the begin of the cycle in nanoseconds.
"

 ::= { lldpXPnoLocEntry 19 }

lldpXPnoLocPortGreenValidity OBJECT-TYPE
SYNTAX TruthValue
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"
The value of this object indicates whether the value of
lldpXPnoLocPortGreenOffset of the according table entry is
valid or not.
"

 ::= { lldpXPnoLocEntry 20 }

-----
-- PNO - Remote System Information
-----

lldpXPnoRemTable OBJECT-TYPE
SYNTAX SEQUENCE OF LldpXPnoRemEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"
This table contains one or more rows per physical network
connection known to this agent. The agent may wish to
ensure that only one lldpXPnoRemEntry is present for
each local port, or it may choose to maintain multiple
lldpXPnoRemEntries for the same local port.
"

 ::= { lldpXPnoRemoteData 1 }

lldpXPnoRemEntry OBJECT-TYPE
SYNTAX LldpXPnoRemEntry
MAX-ACCESS not-accessible
STATUS current
DESCRIPTION
"
Each entry represents the received information of the
communication partner on this physical connection.

The entries feature multiple indices from the
lldpRemEntry of the LLDP-MIB, therefore it is
only present along with the description defined
by the associated lldpRemEntry entry.
"

INDEX { lldpRemTimeMark, lldpRemLocalPortNum, lldpRemIndex }
 ::= { lldpXPnoRemTable 1 }

LldpXPnoRemEntry ::=
    lldpXPnoRemLPDValue      Unsigned32,
    lldpXPnoRemPortTxDValue  Unsigned32,
    lldpXPnoRemPortRxDValue  Unsigned32,
    lldpXPnoRemPortStatusRT2 INTEGER,
    lldpXPnoRemPortStatusRT3 INTEGER,
    lldpXPnoRemPortNoS       DisplayString,
    lldpXPnoRemPortMrpUuid    OCTET STRING,
    lldpXPnoRemPortMrrtStatus INTEGER,
    lldpXPnoRemPortPtcpMaster MacAddress,

```



```

lldpXPnoRemPortPtcpSubdomainUUID OCTET STRING,
lldpXPnoRemPortPtcpIRDataUUID OCTET STRING,
lldpXPnoRemPortModeRT3 INTEGER,
lldpXPnoRemPortPeriodLength Unsigned32,
lldpXPnoRemPortPeriodValidity TruthValue,
lldpXPnoRemPortRedOffset Unsigned32,
lldpXPnoRemPortRedValidity TruthValue,
lldpXPnoRemPortOrangeOffset Unsigned32,
lldpXPnoRemPortOrangeValidity TruthValue,
lldpXPnoRemPortGreenOffset Unsigned32,
lldpXPnoRemPortGreenValidity TruthValue
}

lldpXPnoRemLPDValue OBJECT-TYPE
SYNTAX Unsigned32
UNITS "ns"
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "
    This integer value represents the line propagation
    delay in nanoseconds which was measured by the remote
    system on the corresponding port.

    A value of zero shall be used if the remote system either
    could not accomplish the measurement or does not support
    such a measurement.
    "

 ::= { lldpXPnoRemEntry 1 }

lldpXPnoRemPortTxDValue OBJECT-TYPE
SYNTAX Unsigned32
UNITS "ns"
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "
    This integer value represents the PortTxDelay in
    nanoseconds which was measured by the remote
    system on the corresponding port.

    A value of zero shall be used if the remote system either
    could not accomplish the measurement or does not support
    such a measurement.
    "

 ::= { lldpXPnoRemEntry 2 }

lldpXPnoRemPortRxDValue OBJECT-TYPE
SYNTAX Unsigned32
UNITS "ns"
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "
    This integer value represents the PortRxDelay in
    nanoseconds which was measured by the remote
    system on the corresponding port.

    A value of zero shall be used if the remote system either
    could not accomplish the measurement or does not support
    such a measurement.
    "

 ::= { lldpXPnoRemEntry 3 }

lldpXPnoRemPortStatusRT2 OBJECT-TYPE
SYNTAX INTEGER {
    off(0),
    configured(1),
    running(2)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "
    This value represents the status of the corresponding
    port of the remote system according to RT class 2.

    A value of off(0) means that there isn't any RT2
    capability available for this port. When the port is
    configured for RT2 mode, but the mode isn't active yet
    the value will be configured(1). If the RT2 mode is
    configured for this port and the mode is active, the
    value will be running(2).
    "

 ::= { lldpXPnoRemEntry 4 }

lldpXPnoRemPortStatusRT3 OBJECT-TYPE
SYNTAX INTEGER {
    off(0),
    configured(1),
    up(2),
    down(3),
    running(4)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
    "
    This value represents the status of the corresponding
    port of the remote system according to RT class 3.

    A value of off(0) means that there isn't any RT3
    capability available for this port. When the port is
    configured for RT3 mode, but the mode isn't active yet
    the value will be configured(1).
    When the port is ready for transmission and reception
    of RT3 traffic, the port status will be running(4).
    "

 ::= { lldpXPnoRemEntry 5 }

```

lldpXPnoRemPortNoS OBJECT-TYPE
SYNTAX DisplayString
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The PROFINET NameofStation of the remote partner. If the station isn't configured yet, the value of this object will be the MAC address of the device as a string."
 ::= { lldpXPnoRemEntry 6 }

lldpXPnoRemPortMrpUuid OBJECT-TYPE
SYNTAX OCTET STRING (SIZE (16))
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The UUID of the MRP domain to which the corresponding port of the remote system belongs to. If the port doesn't belong to a MRP domain, the value must be NIL ('0000000000000000')."
 ::= { lldpXPnoRemEntry 7 }

lldpXPnoRemPortMrtrStatus OBJECT-TYPE
SYNTAX INTEGER {
off(0),
configured(1),
up(2)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This object reports the status of the MRRT entity of the corresponding port.
A value of off(0) means that there isn't any MRRT capability available for this port or it is switched off. The value configured(1) indicates that MRRT is configured for the port. When MRRT is active on the port, the value will be up(2)."
 ::= { lldpXPnoRemEntry 8 }

lldpXPnoRemPortPtcpMaster OBJECT-TYPE
SYNTAX MacAddress
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The interface MAC address of the PTCp sync master device of the PTCp subdomain. If unknown it shall be set to zero."
 ::= { lldpXPnoRemEntry 9 }

lldpXPnoRemPortPtcpSubdomainUUID OBJECT-TYPE
SYNTAX OCTET STRING (SIZE (16))
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The UUID of the PTCp subdomain to which this port belongs to. If the port doesn't belong to a PTCp subdomain or the subdomain is invalid or unknown the value of this object must be NIL ('0000000000000000')."
 ::= { lldpXPnoRemEntry 10 }

lldpXPnoRemPortPtcpIRDataUUID OBJECT-TYPE
SYNTAX OCTET STRING (SIZE (16))
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The UUID of the IR data domain to which this port belongs to. If the port doesn't belong to a IR data domain or the domain is invalid or unknown the value of this object must be NIL ('0000000000000000')."
 ::= { lldpXPnoRemEntry 11 }

lldpXPnoRemPortModeRT3 OBJECT-TYPE
SYNTAX INTEGER
{
standard(1),
optimized(0)
}
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"The mode in which the RT3 status is given. If the object is in standard mode all five values of RT3 status are valid, else only ..."
 ::= { lldpXPnoRemEntry 12 }

lldpXPnoRemPortPeriodLength OBJECT-TYPE
SYNTAX Unsigned32 (31250..4000000)
UNITS "ns"
MAX-ACCESS read-only
STATUS current
DESCRIPTION
"This integer value represents the duration of a cycle in nanoseconds on the corresponding port. The value shall be a multiply of 31250 nanoseconds."
 ::= { lldpXPnoRemEntry 13 }

lldpXPnoRemPortPeriodValidity OBJECT-TYPE

```

SYNTAX          TruthValue
MAX-ACCESS      read-only
STATUS          current
DESCRIPTION      "
                The value of this object indicates whether the value of
                lldpXPnoRemPortPeriodLength of the according table entry is
                valid or not.
                "

 ::= { lldpXPnoRemEntry 14 }

lldpXPnoRemPortRedOffset OBJECT-TYPE
SYNTAX              Unsigned32 (0..39999999)
UNITS                "ns"
MAX-ACCESS          read-only
STATUS              current
DESCRIPTION          "
                This integer value represents the begin of the RT_CLASS_3 period
                of the cycle of the receive direction on the corresponding port
                as an offset relative to the begin of the cycle in nanoseconds.
                "

 ::= { lldpXPnoRemEntry 15 }

lldpXPnoRemPortRedValidity OBJECT-TYPE
SYNTAX              TruthValue
MAX-ACCESS          read-only
STATUS              current
DESCRIPTION          "
                The value of this object indicates whether the value of
                lldpXPnoRemPortRedOffset of the according table entry is
                valid or not.
                "

 ::= { lldpXPnoRemEntry 16 }

lldpXPnoRemPortOrangeOffset OBJECT-TYPE
SYNTAX              Unsigned32 (0..39999999)
UNITS                "ns"
MAX-ACCESS          read-only
STATUS              current
DESCRIPTION          "
                This integer value represents the begin of the RT_CLASS_2 period
                of the cycle of the receive direction on the corresponding port
                as an offset relative to the begin of the cycle in nanoseconds.
                "

 ::= { lldpXPnoRemEntry 17 }

lldpXPnoRemPortOrangeValidity OBJECT-TYPE
SYNTAX              TruthValue
MAX-ACCESS          read-only
STATUS              current
DESCRIPTION          "
                The value of this object indicates whether the value of
                lldpXPnoRemPortOrangeOffset of the according table entry is
                valid or not.
                "

 ::= { lldpXPnoRemEntry 18 }

lldpXPnoRemPortGreenOffset OBJECT-TYPE
SYNTAX              Unsigned32 (0..39999999)
UNITS                "ns"
MAX-ACCESS          read-only
STATUS              current
DESCRIPTION          "
                This integer value represents the begin of the unrestricted period
                of the cycle of the receive direction on the corresponding port
                as an offset relative to the begin of the cycle in nanoseconds.
                "

 ::= { lldpXPnoRemEntry 19 }

lldpXPnoRemPortGreenValidity OBJECT-TYPE
SYNTAX              TruthValue
MAX-ACCESS          read-only
STATUS              current
DESCRIPTION          "
                The value of this object indicates whether the value of
                lldpXPnoRemPortGreenOffset of the according table entry is
                valid or not.
                "

 ::= { lldpXPnoRemEntry 20 }

-----
-- Conformance Information
-----

lldpXPnoConformance OBJECT IDENTIFIER ::= { lldpXPnoMIB 2 }
lldpXPnoCompliances OBJECT IDENTIFIER ::= { lldpXPnoConformance 1 }
lldpXPnoGroups       OBJECT IDENTIFIER ::= { lldpXPnoConformance 2 }

-- compliance statements

lldpXPnoCompliance MODULE-COMPLIANCE
STATUS              current
DESCRIPTION          "
                The compliance statement for SNMP entities which
                implement the PNO organizationally defined LLDP
                extension MIB.
                "

MODULE
MANDATORY-GROUPS   { lldpXPnoConfigGroup, lldpXPnoLocGroup, lldpXPnoRemGroup }
GROUP              lldpXPnoMrpGroup
DESCRIPTION          "Required if the system provides MRP."
GROUP              lldpXPnoPtcpGroup
DESCRIPTION          "Required if the system provides PTCP."
 ::= { lldpXPnoCompliances 1 }

-- MIB groupings

```

```

lldpXPnoConfigGroup          OBJECT-GROUP
OBJECTS                      {
                                lldpXPnoConfigSPDTEnable,
                                lldpXPnoConfigPortStatusTxEnable,
                                lldpXPnoConfigAliasTxEnable
                                }
STATUS                        current
DESCRIPTION                   "
                                The collection of objects which are used to configure the
                                PNO organizationally defined LLDP extension implementation behavior.

                                This group is mandatory for agents which implement the PNO
                                organizationally defined LLDP extension, because the information
                                about the signal propagation delay is necessary to configure
                                PROFINET domains.
                                "

::= { lldpXPnoGroups 1 }

lldpXPnoLocGroup             OBJECT-GROUP
OBJECTS                      { lldpXPnoLocLPDValue,
                                lldpXPnoLocPortTxDValue,
                                lldpXPnoLocPortRxDValue,
                                lldpXPnoLocPortStatusRT2,
                                lldpXPnoLocPortStatusRT3,
                                lldpXPnoLocPortNoS,
                                lldpXPnoLocPortModeRT3,
                                lldpXPnoLocPortPeriodLength,
                                lldpXPnoLocPortPeriodValidity,
                                lldpXPnoLocPortRedOffset,
                                lldpXPnoLocPortRedValidity,
                                lldpXPnoLocPortOrangeOffset,
                                lldpXPnoLocPortOrangeValidity,
                                lldpXPnoLocPortGreenOffset,
                                lldpXPnoLocPortGreenValidity
                                }
STATUS                        current
DESCRIPTION                   "
                                The collection of objects which are used to configure the
                                PNO organizationally defined LLDP extension implementation behavior.

                                This group is mandatory for agents which implement the PNO
                                organizationally defined LLDP extension, because the information
                                about the signal propagation delay is necessary to configure
                                PROFINET domains.
                                "

::= { lldpXPnoGroups 2 }

lldpXPnoRemGroup             OBJECT-GROUP
OBJECTS                      { lldpXPnoRemLPDValue,
                                lldpXPnoRemPortTxDValue,
                                lldpXPnoRemPortRxDValue,
                                lldpXPnoRemPortStatusRT2,
                                lldpXPnoRemPortStatusRT3,
                                lldpXPnoRemPortNoS,
                                lldpXPnoRemPortModeRT3,
                                lldpXPnoRemPortPeriodLength,
                                lldpXPnoRemPortPeriodValidity,
                                lldpXPnoRemPortRedOffset,
                                lldpXPnoRemPortRedValidity,
                                lldpXPnoRemPortOrangeOffset,
                                lldpXPnoRemPortOrangeValidity,
                                lldpXPnoRemPortGreenOffset,
                                lldpXPnoRemPortGreenValidity
                                }
STATUS                        current
DESCRIPTION                   "
                                The collection of objects which are used to configure the
                                PNO organizationally defined LLDP extension implementation behavior.

                                This group is mandatory for agents which implement the PNO
                                organizationally defined LLDP extension, because the information
                                about the signal propagation delay is necessary to configure
                                PROFINET domains.
                                "

::= { lldpXPnoGroups 3 }

lldpXPnoMrpGroup             OBJECT-GROUP
OBJECTS                      { lldpXPnoConfigMrpTxEnable,
                                lldpXPnoLocPortMrpUuid,
                                lldpXPnoLocPortMrpStatus,
                                lldpXPnoRemPortMrpUuid,
                                lldpXPnoRemPortMrpStatus
                                }
STATUS                        current
DESCRIPTION                   "
                                The collection of objects which are used to configure the
                                PNO organizationally defined LLDP extension implementation behavior.

                                This group is mandatory for agents which implement the PNO
                                organizationally defined LLDP extension, because the information
                                about the signal propagation delay is necessary to configure
                                PROFINET domains.
                                "

::= { lldpXPnoGroups 4 }

lldpXPnoPtcpGroup            OBJECT-GROUP
OBJECTS                      { lldpXPnoConfigPtcpTxEnable,
                                lldpXPnoLocPortPtcpMaster,
                                lldpXPnoLocPortPtcpSubdomainUUID,
                                lldpXPnoLocPortPtcpIRDataUUID,
                                lldpXPnoRemPortPtcpMaster,
                                lldpXPnoRemPortPtcpSubdomainUUID,
                                lldpXPnoRemPortPtcpIRDataUUID
                                }

```

```

STATUS          current
DESCRIPTION      "
                  The collection of objects which are used to configure the
                  PNO organizationally defined LLDP extension implementation behavior.

                  This group is mandatory for agents which implement the PNO
                  organizationally defined LLDP extension, because the information
                  about the signal propagation delay is necessary to configure
                  PROFINET domains.
                  "

 ::= { lldpXPnoGroups 5 }
END

```

4.16 Common DLL Mapping Protocol Machines

4.16.1 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) and bumpless media redundancy (RRT_Relay).

NOTE Forwarding of standard frames is according to IEEE 802.1D and therefore not specified here.

Mapping to Media Access Control interface (MMAC) and send/receive functions for synchronization messages

NOTE Invoking of MAC is according to IEEE 802.3 and therefore not specified here.

SYNC_SHIMP to add precise time stamps for synchronization

The Interface between the service handler (LMPM), forwarding machines and Mapping to Media Access Control (MMAC) consists of a set of queues and global data.

Figure 47 illustrates the structuring of the protocol machines for the DMPM for a device.

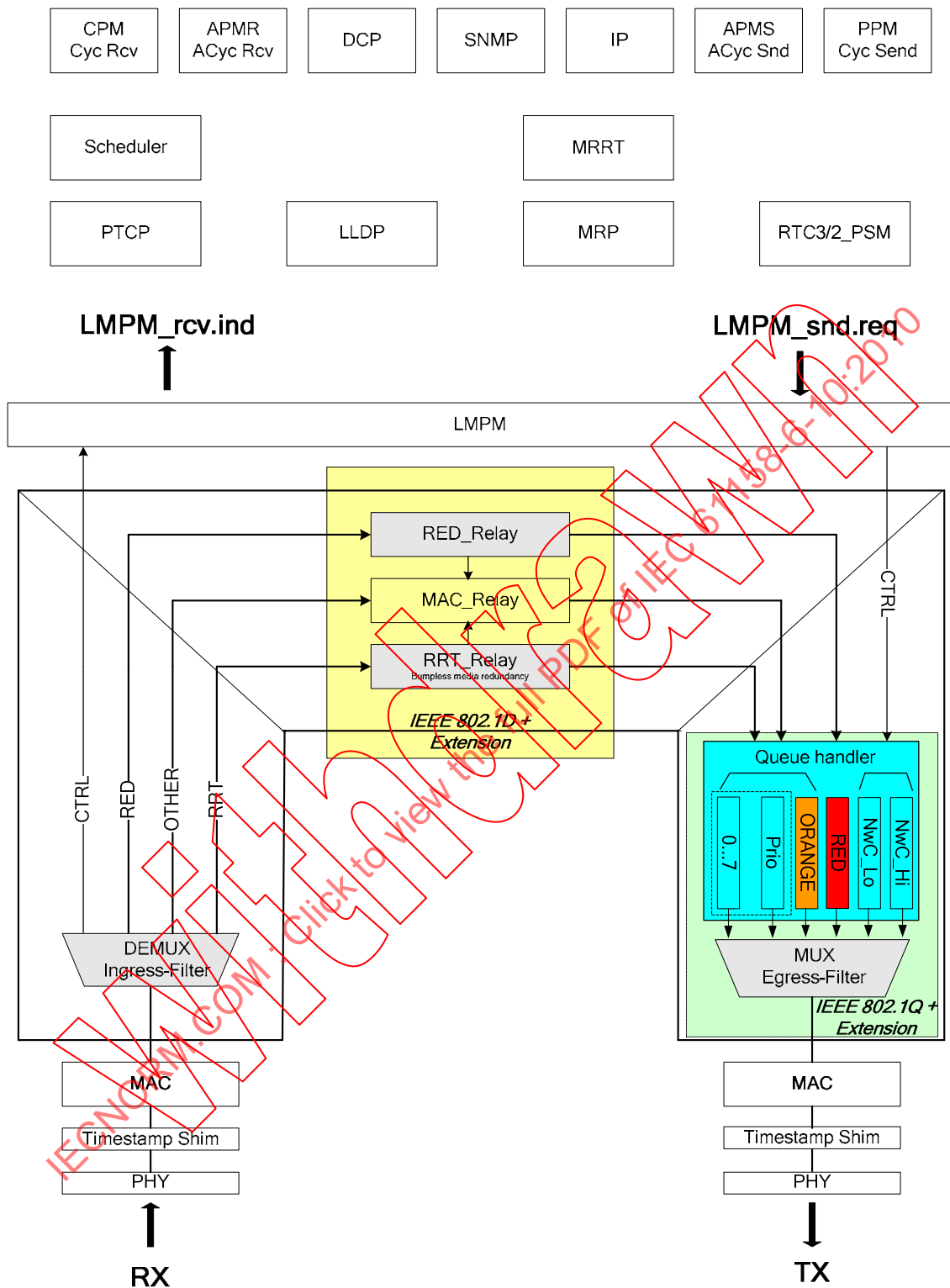


Figure 47 – Structuring of the protocol machines within the DMPM (bridge)

4.16.2 LMPM

4.16.2.1 Primitive definitions

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

4.16.2.2 State machine description

The LMPM forms the interface between MMAC and LMPM-User for various data and management services. The services are all handled in the READY-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 MMAC will be moved from the request queues to the confirmation queues. All valid incoming services will be put into the indication queue.

Local Variables

The LMPM does not use local variables.

4.16.2.3 LMPM state table

The LMPM State Table is shown in Table 238.

Withdrawing
IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 238 – LMPM state table

#	Current State	Event /Condition =>Action	Next State
1	PON	=>	READY
2	READY	LMPM_P_Data.req (CREP, D_Port, Tstamp, DA, SA, N_SDU) /CHECK_PAR => LMPM_status := IV LMPM_P_Data.cnf (CREP, D_Port, TStamp, LMPM_status)	READY
3	READY	LMPM_P_Data.req (CREP, D_Port, TStamp, DA, SA, N_SDU) /CHECK_PAR && !RESOURCE (N_SDU.Len) => LMPM_status := LS LMPM_P_Data.cnf (CREP, D_Port, TStamp, LMPM_status)	READY
4	READY	LMPM_P_Data.req (CREP, D_Port, TStamp, DA, SA, N_SDU) /CHECK_PAR && RESOURCE (N_SDU.Len) => MUX_Data.req (CREP, D_Port, TStamp, N_SDU)	READY
5	READY	MUX_Data.cnf (CREP, D_Port, TStamp, Status) => LMPM_status := Status LMPM_P_Data.cnf (CREP, D_Port, TStamp, LMPM_status)	READY
6	READY	LMPM_N_Data.req (CREP, DA, SA, VLANPrio, VLANID, N_SDU) /CHECK_PAR => LMPM_status := IV LMPM_N_Data.cnf (CREP, LMPM_status)	READY
7	READY	LMPM_N_Data.req (CREP, DA, SA, VLANPrio, VLANID, N_SDU) /CHECK_PAR && !RESOURCE (N_SDU.Len) => LMPM_status := LS LMPM_N_Data.cnf (CREP, LMPM_status)	READY
8	READY	LMPM_N_Data.req (CREP, DA, SA, VLANPrio, VLANID, N_SDU) /CHECK_N_PAR && RESOURCE (N_SDU.Len) => LMPM_status := OK LMPM_N_Data.cnf (CREP, LMPM_status) MAC_RelayData.req (CREP, DA, SA, VLANPrio, VLANID, N_SDU)	READY
9	READY	MAC_RelayData.cnf (CREP, Status) => ignore	READY
10	READY	LMPM_A_Data.req (CREP, DA, SA, VLANPrio, VLANID, A_SDU) /CHECK_A_PAR => LMPM_status := IV LMPM_A_Data.cnf (CREP, LMPM_status)	READY
11	READY	LMPM_A_Data.req (CREP, DA, SA, Prio,VLAN_Tag, LT, A_SDU) /CHECK_A_PAR && !RESOURCE (A_SDU.Len) => LMPM_status := LS LMPM_A_Data.cnf (CREP, LMPM_status)	READY
12	READY	LMPM_A_Data.req (CREP, DA, SA, VLANPrio, VLANID, A_SDU) /CHECK_A_PAR && RESOURCE (A_SDU.Len) => LMPM_status := OK LMPM_A_Data.cnf (CREP, LMPM_status)	READY

#	Current State	Event /Condition =>Action	Next State
		MAC_RelayData.req(CREP, DA, SA, VLANPrio, VLANID, A_SDU)	
13	READY	MAC_RelayData.cnf (CREP, Status) => ignore	READY
14	READY	LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status) /!CHECK_RTC2 !CHECK_RTC3 => LMPM_status := IV LMPM_C_Data.cnf (CREP, LMPM_status)	READY
15	READY	LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status) /CHECK_RTC2 && !RESOURCE (C_SDU.Len) => LMPM_status := LS LMPM_C_Data.cnf (CREP, LMPM_status)	READY
16	READY	LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status) /CHECK_RTC3 && !RESOURCE (C_SDU.Len) => LMPM_status := LS LMPM_C_Data.cnf (CREP, LMPM_status)	READY
17	READY	LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status) /CHECK_RTC2 && RESOURCE (C_SDU.Len) && !MRRT_ON => LMPM_status := OK LMPM_C_Data.cnf (CREP, LMPM_status) MAC_RelayData.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)	READY
18	READY	LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status) /CHECK_RTC2 && RESOURCE (C_SDU.Len) && MRRT_ON => S_Port := LOCAL LMPM_status := OK LMPM_C_Data.cnf (CREP, LMPM_status) RRT_RelayData.req (CREP, S_Port, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)	READY
19	READY	LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status) /CHECK_RTC3 && RESOURCE (C_SDU.Len) => LMPM_status := OK LMPM_C_Data.cnf (CREP, LMPM_status) RED_RelayData.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)	READY
20	READY	MAC_RelayData.cnf (CREP, Status) => ignore	READY
21	READY	RRT_RelayData.cnf (CREP, Status) => ignore	READY
22	READY	RED_RelayData.cnf (CREP, Status) => ignore	READY
23	READY	DMUX_P_Data.ind (CREP, S_Port, TStamp, DA, SA, N_SDU) => LMPM_P_Data.ind (CREP, S_Port, TStamp, DA, SA, N_SDU)	READY

#	Current State	Event /Condition =>Action	Next State
24	READY	RED_RelayData.ind (CREP, DA, SA, LT, VLANPrio, VLANID, C_SDU, APDU_status) => LMPM_C_Data.ind (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)	READY
25	READY	RRT_RelayData.ind (CREP, DA, SA, LT, VLANPrio, VLANID, C_SDU, APDU_status) => LMPM_C_Data.ind (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_status)	READY
26	READY	MAC_RelayData.ind (CREP, DA, SA, LT, VLANPrio, VLANID, C_SDU, APDU_status) /RTC => LMPM_C_Data.ind (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_status)	READY
27	READY	MAC_RelayData.ind (CREP, DA, SA, LT, VLANPrio, VLANID, C_SDU, APDU_status) /RTA => LMPM_A_Data.ind (CREP, DA, SA, VLANPrio, VLANID, A_SDU)	READY
28	READY	MAC_RelayData.ind (CREP, DA, SA, LT, VLANPrio, VLANID, N_SDU) /!RTA => LMPM_N_Data.ind (CREP, DA, SA, VLANPrio, VLANID, N_SDU)	READY
29	READY	LMPM_RED_ScheduleAdd.req (CREP, Port, ReductionRatio, Phase) /Phase == 0 => LMPM_status := PHASE_ERROR LMPM_RED_ScheduleAdd.cnf (CREP, Port, LMPM_status)	READY
30	READY	LMPM_RED_ScheduleAdd.req (CREP, Port, ReductionRatio, Phase) /Phase != 0 && (Phase < ReductionRatio) && (ReductionRatio > 0) && (ReductionRatio <= MAX_REDACTION_RATIO) => RED_RelayScheduleAdd.req (CREP, Port, ReductionRatio, Phase)	READY
31	READY	LMPM_RED_ScheduleAdd.req (CREP, Port, ReductionRatio, Phase) /Phase != 0 && ((Phase >= ReductionRatio) (ReductionRatio == 0) (ReductionRatio > MAX_REDACTION_RATIO)) => LMPM_status := RATIO_ERROR LMPM_RED_ScheduleAdd.cnf (CREP, Port, LMPM_status)	READY
32	READY	RED_RelayScheduleAdd_Cnf (CREP, Port, Status) => LMPM_status := Status LMPM_RED_ScheduleAdd.cnf (CREP, Port, LMPM_status)	READY
33	READY	LMPM_RED_ScheduleRemove.req (CREP, Port, ReductionRatio, Phase) => RED_RelayScheduleRemove.req (CREP, Port, ReductionRatio, Phase)	READY
34	READY	RED_RelayScheduleRem.cnf (CREP, Port, Status) => LMPM_status := Status LMPM_RED_ScheduleRemove.cnf (CREP, Port, LMPM_status)	READY
35	READY	RED_RelayScheduleError.ind(CREP, ErrCode) => LMPM_status := ErrCode LMPM_IRT_Schedule_Error.ind (CREP, LMPM_status)	READY

4.16.2.4 Macros

The LMPM macros are summarized in Table 239.

Table 239 – LMPM macros table

Function name	Operations
RESOURCE	Check that local resources are available to handle the requested data
MRRT_Mode	MRRT is active
CHECK_PAR	Check that all parameters of Service LMPM_N_Data.req or LMPM_P_Data.req are valid. N_SDU.Len: 0 .. 1500 N_SDU.Data: according to DLSDU.Len
CHECK_A_PAR	Check that all parameters of Service LMPM_A_Data.req are valid. A_SDU.LT == RT FrameID within range of RT acyclic A_SDU.Len: 0 .. 1440 A_SDU.Data: according to DLSDU.Len
CHECK_RTC2	Check that all parameters of Service LMPM_C_Data.req are valid. C_SDU.LT == RT FrameID within range of RT Class 2 C_SDU.Len: 0 .. 1440 C_SDU.Data: according to DLSDU.Len
CHECK_RTC3	Check that all parameters of Service LMPM_C_Data.req are valid. Number of parameters must be 7. C_SDU.LT == RT FrameID within range of RT Class 3 C_SDU.Len: 0 .. 1440 C_SDU.Data: according to DLSDU.Len
RTC	Cyclic RT Frame
RTA	Acyclic RT Frame

4.16.2.5 Functions

Table 240 contains the functions used by the LMPM and their descriptions.

Table 240 – LMPM function table

Name	Meaning
LMPM_A_Data	Handle RTA_CLASS_x requests, confirmations and indications and their queuing.
LMPM_N_Data	Handle other requests, confirmations and indications and their queuing.
LMPM_C_Data	Handle RT_CLASS_x requests, confirmations and indications and their queuing.
LMPM_P_Data	Handle Network Control requests, confirmations and indications and their queuing.
LMPM_RED_ScheduleAdd	Handle RT_CLASS_3 requests for each port.
LMPM_RED_ScheduleRemove	Handle RT_CLASS_3 requests for each port.
LMPM_IRT_Schedule_Error	Handle RT_CLASS_3 errors for each port.
MUX_Data	—
RED_RelayScheduleAdd	—
RED_RelayScheduleRemove	—

5 Application layer protocol specification for distributed automation

5.1 FAL syntax description

5.1.1 IDL specification

5.1.1.1 Basic information

The IDL specification provides the formal specification of the FAL service parameters with their data types. This information is used for data marshaling by the ORPC wire protocol to build the APDU. Therefore, the IDL specification describes the format, the order and the content of the APDU. The encoding of the basic data types itself belongs to the used ORPC wire protocol and is part of the underlying abstract ORPC model.

5.1.1.2 Error messages

Table 241 shows the specific error messages (HRESULT data type) for the FAL AE. Error messages are 32 bit values laid out as shown in Figure 48.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
S	R	Facility															Code														

where

S - (1 bit) is the severity code

0 – Success

1 – Error

R - (2 bits) reserved, to be set to zero

Facility - (13 bits) is the facility code

Code - (16 bits) is the facility's status code

Figure 48 – Error message structure

A particular HRESULT value by convention uses the following naming structure:

CBA_<SEV: S or E>_<Reason>.

Table 241 – Error messages

Value	Code	Description
CBA_E_ACCESSBLOCKED	0x8004CB15L	Access to this item is blocked because a constant is currently assigned to it.
CBA_E_CAPACITYEXCEEDED	0x8004CB1DL	The request exceeds the resources according to the machines capacity.
CBA_E_COUNTEXCEEDED	0x8004CB16L	The request exceeds the resources according to their count.
CBA_E_CRDATALENGTH	0x8004CB1BL	The length of the requested data for this CR is invalid for RT.
CBA_E_DEFECT	0x8004CB11L	Hardware defect detected, replacement needed.
CBA_E_DISCONNECTRUNNING	0x8004CB21L	The request exceeds the resources and there is a Disconnect running which will free some resources later.
CBA_E_FLAGUNSUPPORTED	0x8004CB1CL	One of the flags used are not supported in this implementation.
CBA_E_FRAMECOUNTUNSUPPORTED	0x8004CB23L	The number of RT frames transferred between two ACCOs at the same QoS is restricted by the implementation.
CBA_E_INUSE	0x8004CB0EL	The destination specified in the item identifier is already connected to some other provider.
CBA_E_INVALIDCONNECTION	0x8004CB08L	It is not allowed to specify a connection from an identifier to itself.
CBA_E_INVALIDCOOKIE	0x8004CB09L	The Cookie value is not valid.
CBA_E_INVALIDENUMVALUE	0x8004CB04L	The enumeration value is invalid.
CBA_E_INVALIDEPSILON	0x8004CB06L	The Epsilon type or value is not valid.
CBA_E_INVALIDID	0x8004CB05L	The ConsumerID or ProviderID is not valid.
CBA_E_INVALIDSUBSTITUTE	0x8004CB07L	The Substitute type or value is not valid.
CBA_E_ITEMTOOLARGE	0x8004CB1AL	The size of the item exceeds the upper limit for RT.
CBA_E_LIMITVIOLATION	0x8004CB12L	The data exceeds limits of the corresponding data type.
CBA_E_LINKFAILURE	0x8004CB24L	The network cards current link status is unsuitable for this protocol.
CBA_E_LOCATIONCHANGED	0x8004CB22L	The location of the restored connection differs from the location of the saved one.
CBA_E_MALFORMED	0x8004CB00L	The identifier is malformed (too long, unallowed characters, no separation characters, syntactically invalid)
CBA_E_MODECHANGE	0x8004CB25L	The consumer tried to change the data transfer mode from push (OnData-Changed) to pull (GetConnectionData) or vice versa.
CBA_E_NONACCESSIBLE	0x8004CB10L	The item is not accessible.
CBA_E_NOTAPPLICABLE	0x8004CB0FL	The operation is currently not applicable.
CBA_E_NOTROUTABLE	0x8004CB20L	Routing is forbidden for this protocol and the remote station can't be reached via one of the local subnets.
CBA_E_OUTOFACCOPAIRS	0x8004CB19L	The request exceeds the resources according to the AccoPairs.
CBA_E_OUTOFPARTNERACCOS	0x8004CB18L	The request exceeds the resources according to the partner ACCOs.
CBA_E_PERSISTRUNNING	0x8004CB0DL	While the Save service is running, no changes are allowed to the configuration data base (try again in a few seconds).

Value	Code	Description
CBA_E_QCNOTAPPLICABLE	0x8004CB14L	The quality code is not applicable for this operation.
CBA_E_QOSTYPENOTAPPLICABLE	0x8004CB13L	The QoS type is not applicable for this operation.
CBA_E_QOSTYPEUNSUPPORTED	0x8004CB0BL	The QoS type is unsupported.
CBA_E_QOSVALUEUNSUPPORTED	0x8004CB0CL	The QoS value is unsupported.
CBA_E_SIZEEXCEEDED	0x8004CB17L	The request exceeds the resources according to their size.
CBA_E_STATIONFAILURE	0x8004CB1FL	The remote station failed regarding the data transfer.
CBA_E_SUBELEMENTMISMATCH	0x8004CB1EL	The sub element access list does not match the item type.
CBA_E_TIMEVALUEUNSUPPORTED	0x8004CB0AL	The time value is unsupported.
CBA_E_TYPEREMISMATCH	0x8004CB03L	The type specified does not match the expected type.
CBA_E_UNKNOWNMEMBER	0x8004CB02L	The member specified in the item identifier is not known.
CBA_E_UNKNOWNOBJECT	0x8004CB01L	The object specified in the item identifier is not known.
CBA_S_ESTABLISHING	0x0004CA01L	The connection is not yet established.
CBA_S_FRAMEEMPTY	0x0004CA06L	RT Frame is empty, e.g. carries no connections.
CBA_S_NOCONNECTION	0x0004CA02L	There is no connection from the provider to this specific consumer.
CBA_S_NOCONNECTIONDATA	0x0004CA05L	Connection data is currently not received.
CBA_S_PERSISTPENDING	0x0004CA00L	The value requested is currently not persistent.
CBA_S_VALUEBUFFERED	0x0004CA03L	The value was only buffered and has no immediate effect on the process (e.g. device is in state CBAReady).
CBA_S_VALUEUNCERTAIN	0x0004CA04L	It is uncertain, whether the value is valid, since the device is currently in state CBAReady.
DISP_E_BADINDEX	0x8002000BL	Invalid index.
DISP_E_BADPARAMCOUNT	0x8002000EL	The number of elements provided to DISPPARAMS is different from the number of arguments accepted by the method or property.
DISP_E_BADVARTYPE	0x80020008L	One of the arguments is not a valid variant type.
DISP_E_EXCEPTION	0x80020009L	The application needs to raise an exception. In this case, the structure passed in pExcepInfo should be filled in.
DISP_E_MEMBERNOTFOUND	0x80020003L	The requested member does not exist, or the call to Invoke tried to set the value of a read-only property.
DISP_E_NONAMEDARGS	0x80020007L	This implementation of IDispatch does not support named arguments
DISP_E_OVERFLOW	0x8002000AL	One of the arguments could not be coerced to the specified type.
DISP_E_PARAMNOTFOUND	0x80020004L	One of the parameter DISPIDs does not correspond to a parameter on the method. In this case, puArgErr should be set to the first argument that contains the error.
DISP_E_PARAMNOTOPTIONAL	0x8002000FL	A required parameter was omitted.

Value	Code	Description
DISP_E_TYEMISMATCH	0x80020005L	One or more of the arguments could not be coerced. The index within rgvarg of the first parameter with the incorrect type is returned in the puArgErr parameter.
DISP_E_UNKNOWNINTERFACE	0x80020001L	The interface identifier passed in riid is not IID_NULL.
DISP_E_UNKNOWNLCID	0x8002000CL	Unknown language.
DISP_E_UNKNOWNNAME	0x80020006L	Unknown name.
E_ABORT	0x80004004L	
E_ACCESSDENIED	0x80070005L	
E_FAIL	0x80004005L	Unspecified error.
E_HANDLE	0x80070006L	
E_INVALIDARG	0x80070057L	One ore more arguments are invalid.
E_NOINTERFACE	0x80004002L	No such interface supported.
E_NOTIMPL	0x80004001L	The service is not implemented.
E_OUTOFMEMORY	0x8007000EL	Insufficient memory to complete the call.
E_POINTER	0x80004003L	
E_UNEXPECTED	0x8000FFFFL	
RPC_E_INVALID_OBJECT	0x80010114L	The requested object does not exist.
RPC_E_INVALID_OID	0x80070777L	The specified object was not found or recognized.
RPC_E_INVALID_OXID	0x80070776L	The object exporter was not found.
RPC_E_INVALID_SET	0x80070778L	The object exporter set was not found.
RPC_E_VERSION_MISMATCH	0x80010110L	The ORPC version on the client and server machine does not match.
RPC_S_PROCNUM_OUT_OF_RANGE	0x000006D1L	The procedure number is out of range.
S_FALSE	0x00000001L	Success but with additional results, "function worked and the result is false".
S_OK	0x00000000L	Success, "everything worked".
TYPE_E_ELEMENTNOTFOUND	0x8002802BL	Element not found.

5.1.1.3 Interface definitions for the FAL AE

5.1.1.3.1 Overview

The IDL description for the FAL AE, the abstract classes and the used type definitions are defined below. The types are composed of enumerations and structures of other types. These types are used by the FAL services defined in the interface description to store data into a classified and known container or to use quality codes in a readable form. In the class definitions are the interfaces summarized.

5.1.1.3.2 Coding of FAL ASE data types

5.1.1.3.2.1 VARTYPE

The allowed values of VARTYPE are shown in Table 242.

Table 242 – VARTYPE values

Value	Code
VT_EMPTY	0x0000
VT_NULL	0x0001
VT_I2	0x0002
VT_I4	0x0003
VT_R4	0x0004
VT_R8	0x0005
VT_DATE	0x0007
VT_BSTR	0x0008
VT_DISPATCH	0x0009
VT_ERROR	0x000A
VT_BOOL	0x000B
VT_UNKNOWN	0x000D
VT_I1	0x0010
VT_UI1	0x0011
VT_UI2	0x0012
VT_UI4	0x0013
VT_USERDEFINED	0x001D
VT_SAFEARRAY_I2	0x2002
VT_SAFEARRAY_I4	0x2003
VT_SAFEARRAY_R4	0x2004
VT_SAFEARRAY_R8	0x2005
VT_SAFEARRAY_DATE	0x2007
VT_SAFEARRAY_BSTR	0x2008
VT_SAFEARRAY_ERROR	0x200A
VT_SAFEARRAY_BOOL	0x200B
VT_SAFEARRAY_I1	0x2010
VT_SAFEARRAY_UI1	0x2011
VT_SAFEARRAY_UI2	0x2012
VT_SAFEARRAY_UI4	0x2013

5.1.1.3.2.2 ITEMQUALITYDEF

This data type contains the status information of the related data. It consists of three portions: Quality, Substatus and Limits. There are four states of quality (Bad, Uncertain, Good (Non Cascade) and Good (Cascade)), a set of sub-status values for each quality, and four states of the limits (OK, low limited (LL), high limited (HL) and constant (C)). The allowed values are shown in Table 243. The detailed encoding of the quality code is shown in Figure 49.

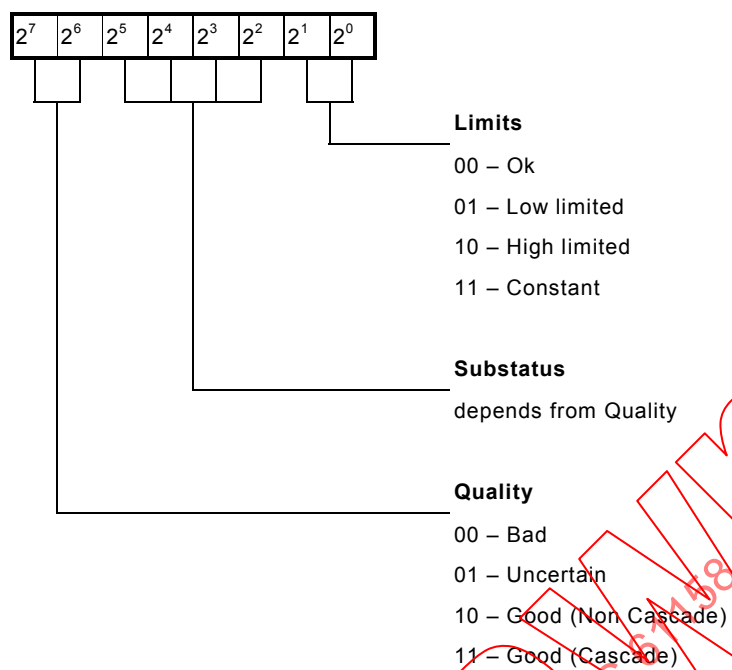


Figure 49 – Coding scheme of ITEMQUALITYDEF

Table 243 – ITEMQUALITYDEF values

Value	Code
BadNonSpecific	0x00
BadNonSpecificLL	0x01
BadNonSpecificHL	0x02
BadNonSpecificC	0x03
BadConfigurationError	0x04
BadConfigurationErrorLL	0x05
BadConfigurationErrorHL	0x06
BadConfigurationErrorC	0x07
BadNotConnected	0x08
BadNotConnectedLL	0x09
BadNotConnectedHL	0x0A
BadNotConnectedC	0x0B
BadDeviceFailure	0x0C
BadDeviceFailureLL	0x0D
BadDeviceFailureHL	0x0E
BadDeviceFailureC	0x0F
BadSensorFailure	0x10
BadSensorFailureLL	0x11
BadSensorFailureHL	0x12
BadSensorFailureC	0x13
BadLastKnownValue	0x14
BadLastKnownValueLL	0x15
BadLastKnownValueHL	0x16
BadLastKnownValueC	0x17
BadCommFailure	0x18
BadCommFailureLL	0x19
BadCommFailureHL	0x1A
BadCommFailureC	0x1B
BadOutOfService	0x1C
BadOutOfServiceLL	0x1D
BadOutOfServiceHL	0x1E
BadOutOfServiceC	0x1F
UncertainNonSpecific	0x40
UncertainNonSpecificLL	0x41
UncertainNonSpecificHL	0x42
UncertainNonSpecificC	0x43
UncertainLastUsableValue	0x44
UncertainLastUsableValueLL	0x45
UncertainLastUsableValueHL	0x46
UncertainLastUsableValueC	0x47
UncertainSubstituteSet	0x48
UncertainSubstituteSetLL	0x49
UncertainSubstituteSetHL	0x4A

Value	Code
UncertainSubstituteSetC	0x4B
UncertainInitialValue	0x4C
UncertainInitialValueLL	0x4D
UncertainInitialValueHL	0x4E
UncertainInitialValueC	0x4F
UncertainSensorNotAccurate	0x50
UncertainSensorNotAccurateLL	0x51
UncertainSensorNotAccurateHL	0x52
UncertainSensorNotAccurateC	0x53
UncertainEngineeringUnitsExceeded	0x54
UncertainEngineeringUnitsExceededLL	0x55
UncertainEngineeringUnitsExceededHL	0x56
UncertainEngineeringUnitsExceededC	0x57
UncertainSubNormal	0x58
UncertainSubNormalLL	0x59
UncertainSubNormalHL	0x5A
UncertainSubNormalC	0x5B
UncertainConfigurationError	0x5C
UncertainConfigurationErrorLL	0x5D
UncertainConfigurationErrorHL	0x5E
UncertainConfigurationErrorC	0x5F
UncertainSimulatedValue	0x60
UncertainSimulatedValueLL	0x61
UncertainSimulatedValueHL	0x62
UncertainSimulatedValueC	0x63
UncertainSensorCalibration	0x64
UncertainSensorCalibrationLL	0x65
UncertainSensorCalibrationHL	0x66
UncertainSensorCalibrationC	0x67
GoodNonCascOk	0x80
GoodNonCascOkC	0x83
GoodNonCascActiveUpdateEvent	0x84
GoodNonCascActiveUpdateEventLL	0x85
GoodNonCascActiveUpdateEventHL	0x86
GoodNonCascActiveUpdateEventC	0x87
GoodNonCascActiveAdvisoryAlarm	0x88
GoodNonCascActiveAdvisoryAlarmLL	0x89
GoodNonCascActiveAdvisoryAlarmHL	0x8A
GoodNonCascActiveAdvisoryAlarmC	0x8B
GoodNonCascActiveCriticalAlarm	0x8C
GoodNonCascActiveCriticalAlarmLL	0x8D
GoodNonCascActiveCriticalAlarmHL	0x8E
GoodNonCascActiveCriticalAlarmC	0x8F
GoodNonCascUnackUpdateEvent	0x90
GoodNonCascUnackUpdateEventLL	0x91

Value	Code
GoodNonCascUnackUpdateEventHL	0x92
GoodNonCascUnackUpdateEventC	0x93
GoodNonCascUnackAdvisoryAlarm	0x94
GoodNonCascUnackAdvisoryAlarmLL	0x95
GoodNonCascUnackAdvisoryAlarmHL	0x96
GoodNonCascUnackAdvisoryAlarmC	0x97
GoodNonCascUnackCriticalAlarm	0x98
GoodNonCascUnackCriticalAlarmLL	0x99
GoodNonCascUnackCriticalAlarmHL	0x9A
GoodNonCascUnackCriticalAlarmC	0x9B
GoodNonCascInitialFailSafe	0xA0
GoodNonCascInitialFailSafeLL	0xA1
GoodNonCascInitialFailSafeHL	0xA2
GoodNonCascInitialFailSafeC	0xA3
GoodNonCascMaintenanceRequired	0xA4
GoodNonCascMaintenanceRequiredLL	0xA5
GoodNonCascMaintenanceRequiredHL	0xA6
GoodNonCascMaintenanceRequiredC	0xA7
GoodCascOk	0xC0
GoodCascOkC	0xC3
GoodCascInitializationAcknowledge	0xC4
GoodCascInitializationAcknowledgeLL	0xC5
GoodCascInitializationAcknowledgeHL	0xC6
GoodCascInitializationAcknowledgeC	0xC7
GoodCascInitializationRequest	0xC8
GoodCascInitializationRequestLL	0xC9
GoodCascInitializationRequestHL	0xCA
GoodCascInitializationRequestC	0xCB
GoodCascNotInvited	0xCC
GoodCascNotInvitedLL	0xCD
GoodCascNotInvitedHL	0xCE
GoodCascNotInvitedC	0xCF
GoodCascDoNotSelect	0xD4
GoodCascDoNotSelectLL	0xD5
GoodCascDoNotSelectHL	0xD6
GoodCascDoNotSelectC	0xD7
GoodCascLocalOverride	0xD8
GoodCascLocalOverrideLL	0xD9
GoodCascLocalOverrideHL	0xDA
GoodCascLocalOverrideC	0xDB
GoodCascInitiateFailSafe	0xE0
GoodCascInitiateFailSafeLL	0xE1
GoodCascInitiateFailSafeHL	0xE2
GoodCascInitiateFailSafeC	0xE3

5.1.1.3.2.3 STATEDEF

The allowed values of STATEDEF are shown in Table 244.

Table 244 – STATEDEF values

Value	Code
CBANonExistent	0x00
CBAInitializing	0x01
CBAReady	0x02
CBAOperating	0x03
CBADefect	0x04

5.1.1.3.2.4 GROUPEXPERDEF

The allowed values of GROUPEXPERDEF are shown in Table 245.

Table 245 – GROUPEXPERDEF values

Value	Code
CBANonAccessible	0x00
CBAOkay	0x01
CBAProblem	0x02
CBAUnknown	0x03

5.1.1.3.2.5 ACCESSRIGHTSDEF

The allowed values of ACCESSRIGHTSDEF are shown in Table 246.

Table 246 – ACCESSRIGHTSDEF values

Value	Code
CBANoAccess	0x00
CBAReadAccess	0x01
CBAWriteAccess	0x02
CBAFullAccess	0x03

5.1.1.3.2.6 PERSISTDEF

The allowed values of PERSISTDEF are shown in Table 247.

Table 247 – PERSISTDEF values

Value	Code
CBAVolatile	0x00
CBAPendingPersistent	0x01
CBAPersistent	0x02

5.1.1.3.2.7 READITEMOUT

```
typedef struct READITEMOUT
{
    VARIANT          Value;           // value
    ITEMQUALITYDEF   QualityCode;     // quality code
}
```

```

        FILETIME      TimeStamp;           // time stamp
        HRESULT        ErrorState;         // partial result
    } READITEMOUT, *pREADITEMOUT;

```

5.1.1.3.2.8 WRITEITEMIN

```

typedef struct WRITEITEMIN
{
    [string]      LPWSTR      Item;           // item name ("Object.Member")
        VARIANT      Value;           // value
} WRITEITEMIN, *pWRITEITEMIN;

```

5.1.1.3.2.9 WRITEITEMQCDIN

```

typedef struct WRITEITEMQCDIN
{
        WRITEITEMIN      WriteItem;           // WRITEITEMIN structure
        ITEMQUALITYDEF    QualityCode;        // quality code
        FILETIME          TimeStamp;          // time stamp
} WRITEITEMQCDIN, *pWRITEITEMQCDIN;

```

5.1.1.3.2.10 ADDCONNECTIONIN

```

typedef struct ADDCONNECTIONIN
{
    [string]      LPWSTR      ProviderItem; // source item name ("Object.Member")
    [string]      LPWSTR      ConsumerItem; // destination item name ("Object.Member")
        PERSISTDEF    Persistence;        // required persistence
        VARIANT        SubstituteValue;    // substitute value
        VARIANT        Epsilon;           // epsilon value (hysteresis)
} ADDCONNECTIONIN, *pADDCONNECTIONIN;

```

5.1.1.3.2.11 ADDCONNECTIONOUT

```

typedef struct ADDCONNECTIONOUT
{
        DWORD          ConsumerID;         // consumer ID
        WORD           Version;            // version number of the connection
        HRESULT        ErrorState;         // partial result
} ADDCONNECTIONOUT, *pADDCONNECTIONOUT;

```

5.1.1.3.2.12 GETIDOUT

```

typedef struct GETIDOUT
{
        DWORD          ConsumerID;         // consumer ID
        BOOLEAN        State;              // activation state (active = TRUE)
        WORD           Version;            // version number
        HRESULT        ErrorState;         // error state of the connection
// A separate HRESULT containing the partial result is not specified,
// because partial errors are not possible here.
} GETIDOUT, *pGETIDOUT;

```

5.1.1.3.2.13 GETCONNECTIONOUT

```

typedef struct GETCONNECTIONOUT
{
    [string]      LPWSTR      Provider;     // provider name ("PDev!LDev")
    [string]      LPWSTR      ProviderItem; // source item name ("Object.Member")
    [string]      LPWSTR      ConsumerItem; // destination item name ("Object.Member")
        VARIANT        SubstituteValue;    // substitute value
        VARIANT        Epsilon;           // epsilon value (hysteresis)
        WORD           QoSType;            // quality of service type
        WORD           QoSValue;           // quality of service value
        BOOLEAN        State;              // activation state (active = TRUE)
        PERSISTDEF    Persistence;        // persistence
        WORD           Version;            // version number
        HRESULT        ErrorState;         // error state of the connection
// A separate HRESULT containing the partial result is not specified.
// That's a bug :- (
// We use this work-around: All partial errors are mapped to E_FAIL. Only
// partial errors cause the result of the GetIDOut method to become S_FALSE.
// The error states of the connections have no influence on the result of
// the method.
} GETCONNECTIONOUT, *pGETCONNECTIONOUT;

```

5.1.1.3.2.14 CONNECTIN

```

typedef struct CONNECTIN

```

```
{
    [string]    LPWSTR        ProviderItem;    // source item name ("Object.Member")
              VARTYPE        DataType;        // basic data type
              VARIANT        Epsilon;        // epsilon value (hysteresis)
              DWORD          ConsumerID;      // consumer ID
} CONNECTIN, *pCONNECTIN;
```

5.1.1.3.2.15 CONNECTOUT

```
typedef struct CONNECTOUT
{
    DWORD        ProviderID;    // provider ID
    HRESULT       ErrorState;    // partial result
} CONNECTOUT, *pCONNECTOUT;
```

5.1.1.3.2.16 GETCONSCONNOUT

```
typedef struct GETCONSCONNOUT
{
    [string]    LPWSTR        Provider;        // provider name ("PDev!LDev")
    [string]    LPWSTR        ProviderItem;    // source item name ("Object.Member")
    [string]    LPWSTR        ConsumerItem;    // destination item name ("Object.Member")
    VARIANT     SubstituteValue;    // substitute value
    VARIANT     Epsilon;        // epsilon value (hysteresis)
    WORD        QoSType;        // quality of service type
    WORD        QoSValue;      // quality of service value
    BOOLEAN     State;        // activation state (active = TRUE)
    PERSISTDEF  Persistence;    // persistence
    HRESULT     PartialResult;    // partial result
} GETCONSCONNOUT, *pGETCONSCONNOUT;
```

5.1.1.3.2.17 DIAGCONSCONNOUT

```
typedef struct DIAGCONSCONNOUT
{
    // rule of thumb: iMap uses GetConsConnections on each AccoStamp change
    // and DiagConsConnections on each GroupError event, so GetConsConnections
    // should include all values which effect an AccoStamp change and DiagCons-
    // Connections should include all values which effect a GroupError change.
    BOOLEAN     State;        // activation state (active = TRUE)
    PERSISTDEF  Persistence;    // persistence
    WORD        Version;      // version number
    HRESULT     ErrorState;    // error state of the connection
    HRESULT     PartialResult;    // partial result
} DIAGCONSCONNOUT, *pDIAGCONSCONNOUT;
```

5.1.1.3.2.18 GETPROVCONNOUT

```
typedef struct GETPROVCONNOUT
{
    [string]    LPWSTR        Consumer;        // consumer name ("PDev!LDev")
    [string]    LPWSTR        ProviderItem;    // source item name ("Object.Member")
    DWORD       ConsumerID;    // consumer ID
    VARIANT     Epsilon;        // epsilon value (hysteresis) - DCOM only
    WORD        QoSType;        // quality of service type
    WORD        QoSValue;      // quality of service value
    BOOLEAN     State;        // activation state (active = TRUE)
    HRESULT     PartialResult;    // partial result
} GETPROVCONNOUT, *pGETPROVCONNOUT;
```

5.1.1.3.2.19 CONNECTIN2

```
typedef struct CONNECTIN2
{
    [string]    LPWSTR        ProviderItem;    // source item name ("Object.Member")
    WORD        TypeDescLen;    // length of type description
    [size_is(TypeDescLen)] WORD *pTypeDesc;    // type description
    VARIANT     Epsilon;        // epsilon value (hysteresis)
    DWORD       ConsumerID;    // consumer ID
} CONNECTIN2, *pCONNECTIN2;
```

5.1.1.3.2.20 MACAddr

```
typedef struct MACAddr
{
    // 6 byte Ethernet MAC address (highest byte first)
    BYTE        B0;
    BYTE        B1;
```

```

        BYTE          B2;
        BYTE          B3;
        BYTE          B4;
        BYTE          B5;
    } MACAddr, *pMACAddr;

```

5.1.1.3.2.21 CONNECTINCR

```

typedef struct CONNECTINCR
{
    WORD          ConsumerCRID;        // frame ID of the CR
    WORD          ConsumerCRLength;    // maximum length of the CR
} CONNECTINCR, *pCONNECTINCR;

```

5.1.1.3.2.22 CONNECTOUTCR

```

typedef struct CONNECTOUTCR
{
    DWORD          ProviderCRID;        // provider CR ID
    HRESULT         PartialResult;      // partial result
} CONNECTOUTCR, *pCONNECTOUTCR;

```

5.1.1.3.2.23 CONNECTINSRT

```

typedef struct CONNECTINSRT
{
    [string]       LPWSTR               ProviderItem;    // source item name ("Object.Member")
    WORD           TypeDescLen;         // length of type description
    [size_is(TypeDescLen)] WORD *pTypeDesc;            // type description
    DWORD          ConsumerID;          // consumer ID
    WORD           Length;              // marshalled length of record
    (redundant information for robustness)
} CONNECTINSRT, *pCONNECTINSRT;

```

5.1.1.3.2.24 UUID

Predefined values are shown in Table 248.

Table 248 – UUID values

Value	Code
UUID_NULL	00000000-0000-0000-0000-000000000000
UUID_IUnknown	00000000-0000-0000-C000-000000000046
UUID_IDispatch	00020400-0000-0000-C000-000000000046
UUID_ICBAPhysicalDevice	CBA00001-6C97-11D1-8271-00A02442DF7D
UUID_ICBAPhysicalDevice2	CBA00006-6C97-11D1-8271-00A02442DF7D
UUID_ICBABrowse	CBA00002-6C97-11D1-8271-00A02442DF7D
UUID_ICBABrowse2	CBA00007-6C97-11D1-8271-00A02442DF7D
UUID_ICBAPersist	CBA00005-6C97-11D1-8271-00A02442DF7D
UUID_ICBAPersist2	CBA00008-6C97-11D1-8271-00A02442DF7D
UUID_ICBALogicalDevice	CBA00011-6C97-11D1-8271-00A02442DF7D
UUID_ICBALogicalDevice2	CBA00017-6C97-11D1-8271-00A02442DF7D
UUID_ICBAState	CBA00012-6C97-11D1-8271-00A02442DF7D
UUID_ICBATime	CBA00014-6C97-11D1-8271-00A02442DF7D
UUID_ICBAGroupError	CBA00015-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoMgt	CBA00041-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoMgt2	CBA00046-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoServer	CBA00043-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoServer2	CBA00048-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoServerSRT	CBA00045-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoCallback	CBA00042-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoCallback2	CBA00047-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoSync	CBA00044-6C97-11D1-8271-00A02442DF7D
UUID_ICBARTAuto	CBA00051-6C97-11D1-8271-00A02442DF7D
UUID_ICBARTAuto2	CBA00052-6C97-11D1-8271-00A02442DF7D
UUID_ICBASystemProperties	CBA00062-6C97-11D1-8271-00A02442DF7D
UUID_PhysicalDevice	CBA00000-6C97-11D1-8271-00A02442DF7D
UUID_LogicalDevice	CBA00010-6C97-11D1-8271-00A02442DF7D
UUID_ACCO	CBA00040-6C97-11D1-8271-00A02442DF7D
UUID_RTAuto	CBA00050-6C97-11D1-8271-00A02442DF7D
UUID_SystemRTAuto	CBA00090-6C97-11D1-8271-00A02442DF7D

5.1.1.3.3 FAL ASE class interfaces

5.1.1.3.3.1 ICBAPhysicalDevice

```
[
    object,
    uuid(CBA00001-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAPhysicalDevice Interface"),
    pointer_default(unique)
]
interface ICBAPhysicalDevice : IDispatch
{
    [propget, helpstring("Producer/manufacturer of the physical device")]
    HRESULT Producer([out, retval] BSTR *pVal);

    [propget, helpstring("Catalog name of the physical device")]
    HRESULT Product([out, retval] BSTR *pVal);

    [propget, helpstring("Serial number of the physical device")]
```

```

        HRESULT SerialNo([out, retval] VARIANT *pVal);

[propget, helpstring("Production date of the physical device")]
        HRESULT ProductionDate([out, retval] DATE *pVal);

[helpstring("Revision major.minor number of physical device")]
        HRESULT Revision(
            [out] SHORT *pMajor,
            [out] SHORT *pMinor);

[propget, helpstring("Return logical device by name")]
        HRESULT LogicalDevice(
            [in] BSTR Name,
            [out, retval] ICBALogicalDevice **ppLDev);
};

```

5.1.1.3.3.2 ICBAPhysicalDevice2

```

[
    object,
    uuid(CBA00006-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAPhysicalDevice2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAPhysicalDevice2 : ICBAPhysicalDevice
{
    [helpstring("Implementation type")]
        HRESULT Type(
            [out] VARIANT_BOOL *pMultiApp,
            [out] VARIANT_BOOL *pPROFINetDCOMStack);

    [helpstring("Revision of PROFINET runtime source")]
        HRESULT PROFINetRevision(
            [out] SHORT *pMajor,
            [out] SHORT *pMinor,
            [out] SHORT *pServicePack,
            [out] SHORT *pBuild);

    [propget, helpstring("Return PDev stamp")]
        HRESULT PDevStamp([out, retval] LONG *pVal);
};

```

5.1.1.3.3.3 ICBABrowse

```

[
    object,
    uuid(CBA00002-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBABrowse Interface"),
    pointer_default(unique)
]
interface ICBABrowse : IDispatch
{
    [propget, helpstring("Return number of items")]
        HRESULT Count([out, retval] LONG *pVal);

    [helpstring("Browse all available items")]
        HRESULT BrowseItems(
            [in] LONG Offset, // offset - !=0 if last call did
                                // not return all items
            [in] LONG MaxReturn, // at most return that many items
            [out] VARIANT *pItem, // SAFEARRAY(BSTR) - item names
            [out, optional] VARIANT *pDataTypes,
            [out, optional] VARIANT *pAccessRights);

    // BrowseItems usage:
    //
    // on PhysicalDevice object:
    //   pItem - names of the LogicalDevices
    //   pDataTypes and pAccessRights - empty
    //
    // on LogicalDevice object:
    //   pItem - names of the non-system RTAutos

```

```

    //    pDataType and pAccessRight - empty
    //
    // on RTAuto object:
    //    pItem - names of the items
    //    pDataType - SAFEARRAY of LONG (actually VARTYPE), containing the
    //               stage 1 type description of the item
    //    pAccessRight - SAFEARRAY of LONG (actually ACCESSRIGHTSDEF), con-
    //               taining the access rights of the item
};

```

5.1.1.3.3.4 ICBABrowse2

```

[
    object,
    uuid(CBA00007-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBABrowse2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBABrowse2 : ICBABrowse
{
    [propget, helpstring("Return number of items")]
    HRESULT Count2(
        [in]          LONG      Selector, // selects things to count
        [out, retval] LONG      *pVal);    // count

    [helpstring("Browse all available items")]
    HRESULT BrowseItems2(
        [in]          LONG      Selector, // selects things to browse
        [in]          LONG      Offset,   // offset - !=0 if last call did
                                         // not return all items
        [in]          LONG      MaxReturn, // at most return that many items
        [out]          VARIANT *pItem,    // SAFEARRAY(BSTR) - item names
        [out, optional] VARIANT *pInfo1,  // Additional info 1
        [out, optional] VARIANT *pInfo2); // Additional info 2

    // BrowseItems2 usage:
    //
    // on PhysicalDevice object (Selector =0):
    //    pItem - names of the LogicalDevices
    //    pInfo1 and pInfo2 - empty
    //
    // on LogicalDevice object (Selector =0):
    //    pItem - names of the non-system RTAutos
    //    pInfo1 and pInfo2 - empty
    //
    // on LogicalDevice object (Selector =1):
    //    pItem - names of the system RTAutos (starting with "!")
    //    pInfo1 and pInfo2 - empty
    //
    // on RTAuto object (Selector =0):
    //    pItem - names of the items
    //    pInfo1 - SAFEARRAY of BSTR (actually, the BSTR must be interpreted
    //           as an ARRAY of SHORT), containing the stage 2 type description of
    //           the item
    //    pInfo2 - SAFEARRAY of LONG (actually ACCESSRIGHTSDEF), containing
    //           the access rights of the item
};

```

5.1.1.3.3.5 ICBAPersist

```

[
    object,
    uuid(CBA00005-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAPersist Interface"),
    pointer_default(unique)
]
interface ICBAPersist : IDispatch
{
    [helpstring("Save all persistent data")]
    HRESULT Save();
};

```

5.1.1.3.3.6 ICBAPersist2

```
[
    object,
    uuid(CBA00008-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAPersist2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAPersist2 : ICBAPersist
{
    [helpstring("Save all persistent data")]
    HRESULT Save2(
        [out] VARIANT *pLDevName,          // SAFEARRAY(BSTR) - LDev names
        [out] VARIANT *pPartialResult);    // SAFEARRAY(LONG) - HRESULT partial
                                          // result
};
```

5.1.1.3.3.7 ICBALogicalDevice

```
[
    object,
    uuid(CBA00011-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBALogicalDevice Interface"),
    pointer_default(unique)
]
interface ICBALogicalDevice : IDispatch
{
    [propget, helpstring("Name of the logical device")]
    HRESULT Name([out, retval] BSTR *pVal);

    [propget, helpstring("Producer/manufacture of the logical device")]
    HRESULT Producer([out, retval] BSTR *pVal);

    [propget, helpstring("Catalog name of the logical device")]
    HRESULT Product([out, retval] BSTR *pVal);

    [propget, helpstring("Serial number of the logical device")]
    HRESULT SerialNo([out, retval] VARIANT *pVal);

    [propget, helpstring("Production date of the logical device")]
    HRESULT ProductionDate([out, retval] DATE *pVal);

    [helpstring("Revision major.minor number of logical device")]
    HRESULT Revision(
        [out] SHORT *pMajor,
        [out] SHORT *pMinor);

    [propget, helpstring("Return pointer to ACCO")]
    HRESULT ACCO([out, retval] IUnknown **ppACCO);

    [propget, helpstring("Return RTAuto by name")]
    HRESULT RTAuto(
        [in] BSTR Name,
        [out, retval] ICBARTAuto **ppAuto);
};
```

5.1.1.3.3.8 ICBALogicalDevice2

```
[
    object,
    uuid(CBA00017-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBALogicalDevice2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBALogicalDevice2 : ICBALogicalDevice
{
    [helpstring("Revision of PROFINET runtime source")]
    HRESULT PROFINetRevision(
        [out] SHORT *pMajor,
        [out] SHORT *pMinor,
        [out] SHORT *pServicePack,
```

```

        [out] SHORT *pBuild);

[helpstring("Component information")]
    HRESULT ComponentInfo(
        [out] BSTR *pComponentID, // GUID as
                                   // "{7005c200-6C97-11D1-8271-00A02442DF7D}"
        [out] BSTR *pVersion);    // version
};

```

5.1.1.3.3.9 ICBAState

```

[
    object,
    uuid(CBA00012-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAState Interface"),
    pointer_default(unique)
]
interface ICBAState : IDispatch
{
    [propget, helpstring("State of the device")]
        HRESULT State([out, retval] STATEDEF *pVal);

    [helpstring("Activate logical device - transfer to state CBAOperating")]
        HRESULT Activate();

    [helpstring("Deactivate logical device - transfer to state CBAReady")]
        HRESULT Deactivate();

    [helpstring("Reset logical device - transfer via state CBAInitializing to state CBAReady")]
        HRESULT Reset();

    [helpstring("Advise of the ICBAStateEvent")]
        HRESULT AdviseState(
            [in] ICBAStateEvent *pStateEvent,
            [out] LONG *pCookie);

    [helpstring("Unadvise of the ICBAStateEvent")]
        HRESULT UnadviseState([in] LONG Cookie);
};

```

5.1.1.3.3.10 ICBATime

```

[
    object,
    uuid(CBA00014-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBATime Interface"),
    pointer_default(unique)
]
interface ICBATime : IDispatch
{
    [propget, helpstring("Get current time")]
        HRESULT Time([out, retval] DATE *pVal);

    [propput, helpstring("Set time")]
        HRESULT Time([in] DATE NewVal);
};

```

5.1.1.3.3.11 ICBAGroupError

```

[
    object,
    uuid(CBA00015-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAGroupError Interface"),
    pointer_default(unique)
]
interface ICBAGroupError : IDispatch
{
    [helpstring("Get current error state")]
        HRESULT GroupError(
            [out] GROUPERRORDEF *pVal,
            [out] LONG *pMagicCookie);
};

```

5.1.1.3.3.12 ICBAAccoMgt

[illegible]

```

[propget, helpstring("Get current ping factor")]
    HRESULT PingFactor([out, retval] WORD *pVal);

[propput, helpstring("Set ping factor")]
    HRESULT PingFactor([in] WORD NewVal);

[propget, helpstring("Return configuration data base cookie")]
    HRESULT CDBCcookie([out, retval] LONG *pVal);
};

```

5.1.1.3.3.13 ICBAAccoMgt2

```

[
    object,
    uuid(CBA00046-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoMgt2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAAccoMgt2 : ICBAAccoMgt
{
    [helpstring("Get current consumer connection IDs")]
    HRESULT GetConsIDs(
        [out]          DWORD *pCount,          // number of connections
        [out, size_is(, *pCount)] DWORD **ppConsumerID; // consumer IDs

    [helpstring("Get consumer connection information")]
    HRESULT GetConsConnections(
        [in]          DWORD Count,          // number of connections
        [in, size_is(Count)] DWORD *pConsumerID, // consumer IDs
        [out, size_is(, Count)] GETCONSCONNOUT **ppGetConsConnOut;
                                                // connection info

    [helpstring("Get consumer connection diagnosis")]
    HRESULT DiagConsConnections(
        [in]          DWORD Count,          // number of connections
        [in, size_is(Count)] DWORD *pConsumerID, // consumer IDs
        [out, size_is(, Count)] DIAGCONSCONNOUT **ppDiagConsConnOut;
                                                // diagnosis info

    [helpstring("Get current provider IDs")]
    HRESULT GetProvIDs(
        [out]          DWORD *pCount,          // number of connections
        [out, size_is(, *pCount)] DWORD **ppProviderID; // provider IDs

    [helpstring("Get provider connection information")]
    HRESULT GetProvConnections(
        [in]          DWORD Count,          // number of connections
        [in, size_is(Count)] DWORD *pProviderID, // provider IDs
        [out, size_is(, Count)] GETPROVCONNOUT **ppGetProvConnOut;
                                                // connection info

    [helpstring("Get ACCO diagnosis")]
    HRESULT GetDiagnosis(
        [in]          DWORD Request,          // request
        [in]          DWORD InLength,          // length of InBuffer
        [in, size_is(InLength)] BYTE *pInBuffer, // InBuffer
        [out]          DWORD *pOutLength,          // length of OutBuffer
        [out, size_is(, *pOutLength)] BYTE **ppOutBuffer; // OutBuffer
};

```

5.1.1.3.3.14 ICBAAccoCallback

```

[
    object,
    uuid(CBA00042-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoCallback Interface"),
    pointer_default(unique)
]
interface ICBAAccoCallback : IUnknown
{
    [helpstring("Hand over new connection data")]
    HRESULT OnDataChanged(
        [in]          long Length,          // buffer length
        [in, size_is(Length)] unsigned char *pBuffer; // buffer
};

```



```
// structure of pBuffer[]:
//
// header:
// 00 UCHAR      version      -- constant=01
// 01 UCHAR      flags
// 02 USHORT     cnt          -- number of records
// 04 array of record
//
// each record:
// 00 USHORT     len          -- length of record
// 02 ULONG      ConsumerID   -- consumer side of connection
// 06 UCHAR      QualityCode  -- OPC QualityCode
// 07           value        -- according to data type
// len-7         -- optional TimeStamp
//
// case ARRAY:
// 07 USHORT     dim          -- array dimensions
// 09 ULONG[dim] elem        -- number of elements in each dimension
// 4*dim+09 etype[elem]     -- elements
//
// case BSTR:
// 07 ULONG      strlen      -- string length (number of wide chars)
// 11 USHORT[strlen] str    -- wide char string (not terminated with 0)
//
// case STRUCT: (stage 2)
// 07 sequence of values, unaligned, depth-first
//
// in ARRAY or STRUCT:
// sequence of values, unaligned, depth-first
// representation: as above, with the following exceptions:
// ARRAY: dimensions and number of elements fields are not present
// BSTR: is filled with zeroes up to the maximum length for fixed layout
};
```

5.1.1.3.3.15 ICBAAccoCallback2

```
[
    object,
    uuid(CBA00047-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoCallback2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAAccoCallback2 : ICBAAccoCallback
{
    [helpstring("Reverse Ping from provider to consumer")]
    HRESULT Gnip();
};
```

5.1.1.3.3.16 ICBAAccoServer

```
[
    object,
    uuid(CBA00043-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoServer Interface"),
    pointer_default(unique)
]
interface ICBAAccoServer : IUnknown
{
    [helpstring("Establish a set of connections")]
    HRESULT Connect(
        [in, string] LPWSTR Consumer,      // consumer name ("PDev!LDev")
        [in]         WORD QoSType,         // common quality of service type
        [in]         WORD QoSValue,        // common quality of service value
        [in]         BOOLEAN State,        // activation state (active = TRUE)
        [in]         ICBAAccoCallback *pICBAAccoCallback, // interface pointer
                                                    // to callback of the consumer
        [in]         DWORD Count,          // number of connections
        [in, size_is(Count)] CONNECTIN *pConnectIn, // connection information
        [out]         BOOLEAN *pFirstConnect, // first connect call
        [out, size_is(,Count)] CONNECTOUT **ppConnectOut); // results

    [helpstring("Purge a set of connections")]
    HRESULT Disconnect(
        [in]         DWORD Count,          // number of connections
```



```

[in, size_is(Count)] DWORD *pProviderID, // provider IDs
[out, size_is(,Count)] HRESULT **ppError); // partial results

[helpstring("Purge all connections of a specific consumer")]
HRESULT DisconnectMe([in, string] LPWSTR Consumer);
// consumer name ("PDev!LDev")

[helpstring("Modify activation state of a set of connections")]
HRESULT SetActivation(
    [in]          BOOLEAN State, // activation state (active = TRUE)
    [in]          DWORD Count, // number of connections
    [in, size_is(Count)] DWORD *pProviderID, // provider IDs
    [out, size_is(,Count)] HRESULT **ppError); // partial results

[helpstring("Lifebeat handling")]
HRESULT Ping([in, string] LPWSTR Consumer); // consumer name ("PDev!LDev")
};

```

5.1.1.3.3.17 ICBAAccoServer2

```

[
    object,
    uuid(CBA00048-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoServer2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAAccoServer2 : ICBAAccoServer
{
    [helpstring("Establish a set of connections")]
    HRESULT Connect2(
        [in, string] LPWSTR Consumer, // consumer name ("PDev!LDev")
        [in]          WORD QoSType, // common quality of service type
        [in]          WORD QoSValue, // common quality of service value
        [in]          BOOLEAN State, // activation state (active = TRUE)
        [in]          ICBAAccoCallback2 *pICBAAccoCallback, // interface
        // pointer to callback of the consumer
        [in]          DWORD Count, // number of connections
        [in, size_is(Count)] CONNECTIN2 *pConnectIn, // connection information
        [out]          BOOLEAN *pFirstConnect, // first connect call
        [out, size_is(,Count)] CONNECTOUT **ppConnectOut); // results

    [helpstring("Get the connection data")]
    HRESULT GetConnectionData(
        [in, string] LPWSTR Consumer, // consumer name ("PDev!LDev")
        [out]          long *pLength, // buffer length
        [out, size_is(,pLength)] unsigned char **ppBuffer); // buffer
    // structure of pBuffer[]:
    // see ICBAAccoCallback::OnDataChanged()
};

```

5.1.1.3.3.18 ICBAAccoServerSRT

```

[
    object,
    uuid(CBA00045-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoServerSRT Interface"),
    pointer_default(unique)
]
// stage 2 interface
interface ICBAAccoServerSRT : IUnknown
{
    [helpstring("Connect a set of communication relations")]
    HRESULT ConnectCR(
        [in, string] LPWSTR Consumer, // consumer name ("PDev!LDev")
        [in]          WORD QoSType, // RT type
        [in]          WORD QoSValue, // value of RT
        [in]          ICBAAccoCallback2 *pICBAAccoCallback2, // interface
        // pointer to the consumer's
        // call-back to allow gnetting.
        [in]          MACAddr ConsumerMAC, // MAC address of consumer
        [in]          DWORD Flags, // flags
        [in]          DWORD Count, // number of CRs to connect
        [in, size_is(Count)] CONNECTINCR *pConnectIn, // CR information
        [out]          BOOLEAN *pFirstConnect, // if true, AR was established
    );
};

```

```

// with this call
[out]          MACAddr *pProviderMAC, // MAC address of provider
[out, size_is(,Count)] CONNECTOUTCR **ppConnectOut); // CR result

[helpstring("Disconnect a communication relation")]
HRESULT DisconnectCR(
    [in]          DWORD Count, // number of CRs to disconnect
    [in, size_is(Count)] DWORD *pProviderCRID, // provider CR IDs
    [out, size_is(,Count)] HRESULT **ppError); // partial results

[helpstring("Establish a set of connections")]
HRESULT Connect(
    [in]          DWORD ProviderCRID, // provider CR ID
    [in]          BOOLEAN State, // activation state (active = TRUE)
    [in]          BOOLEAN LastConnect, // CR is complete, start to
    // provide now
    [in]          DWORD Count, // count of items to connect
    [in, size_is(Count)] CONNECTINSRT *pConnectIn, // connection information
    [out, size_is(,Count)] CONNECTOUT **ppConnectOut); // connection result

[helpstring("Purge a set of connections")]
HRESULT Disconnect(
    [in]          DWORD Count, // number of connections
    [in, size_is(Count)] DWORD *pProviderID, // provider IDs
    [out, size_is(,Count)] HRESULT **ppError); // partial results

[helpstring("Purge all connections of a specific consumer")]
HRESULT DisconnectMe([in, string] LPWSTR Consumer);
// consumer name ("PDev!LDev")

[helpstring("Modify activation state of a set of connections")]
HRESULT SetActivation(
    [in]          BOOLEAN State, // activation state (active = TRUE)
    [in]          DWORD Count, // number of connections
    [in, size_is(Count)] DWORD *pProviderID, // provider IDs
    [out, size_is(,Count)] HRESULT **ppError); // partial results
}

```

5.1.1.3.3.19 ICBAAccoSync

```

[
    object,
    uuid(CBA00044-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoSync Interface"),
    pointer_default(unique)
]
interface ICBAAccoSync : IUnknown
{
    [helpstring("Read items")]
    HRESULT ReadItems(
        [in]          DWORD Count, // number of items to read
        [in, size_is(Count), string] LPWSTR *pReadItem, // items to read
        [out, size_is(,Count)] READITEMOUT **ppReadItemOut);
    // item information

    [helpstring("Write items")]
    HRESULT WriteItems(
        [in]          DWORD Count, // number of items to write
        [in, size_is(Count)] WRITEITEMIN *pWriteItemIn, // items to write
        [out, size_is(,Count)] HRESULT **ppError); // partial results

    [helpstring("Write items with quality code and time stamps")]
    HRESULT WriteItemsQCD(
        [in]          DWORD Count, // number of items to write
        [in, size_is(Count)] WRITEITEMQCDIN *pWriteItemQcdIn,
    // items to write
        [out, size_is(,Count)] HRESULT **ppError); // partial results
};

```

5.1.1.3.3.20 ICBARTAuto

```

[
    object,

```

```

        uuid(CBA00051-6C97-11D1-8271-00A02442DF7D),
        dual, oleautomation,
        helpstring("ICBARTAuto Interface"),
        pointer_default(unique)
    ]
interface ICBARTAuto : IDispatch
{
    [propget, helpstring("Name of the runtime automation object")]
    HRESULT Name([out, retval] BSTR *pVal);

    [helpstring("Revision major.minor number of the runtime automation object")]
    HRESULT Revision(
        [out] SHORT *pMajor,
        [out] SHORT *pMinor);
};

```

5.1.1.3.3.21 ICBARTAuto2

```

[
    object,
    uuid(CBA00052-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBARTAuto2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBARTAuto2 : ICBARTAuto
{
    [helpstring("Component information")]
    HRESULT ComponentInfo(
        [out] BSTR *pComponentID, // GUID as
        // "{7805c200-6C97-11D1-8271-00A02442DF7D}"
        [out] BSTR *pVersion); // version
};

```

5.1.1.3.3.22 ICBASystemProperties

```

[
    object,
    uuid(CBA00062-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBASystemProperties Interface"),
    pointer_default(unique)
]
// stage 2 interface
interface ICBASystemProperties : IDispatch
{
    [propget, helpstring("State collection")]
    HRESULT StateCollection([out, retval] SAFEARRAY(BYTE) *pVal);
    // the SAFEARRAY contains:
    // extended type description (VT_USERDEFINED,5,VT_UI2,VT_UI2,VT_UI2,VT_UI2,
    // VT_UI2)
    // typedef struct STATECOLLECTION
    // {
    //     WORD LDevState;                LDev's operating state (STATEDEF)
    //     WORD LDevGroupError;           LDev's group error (GROUPERRORDEF)
    //     WORD LDevGECookie;             LDev's group error cookie
    //     WORD AccoGroupError;           Acco's group error (GROUPERRORDEF)
    //     WORD AccoGECookie;             Acco's group error cookie
    // } STATECOLLECTION;

    [propget, helpstring("Stamp collection")]
    HRESULT StampCollection([out, retval] SAFEARRAY(BYTE) *pVal);
    // the SAFEARRAY contains:
    // extended type description (VT_USERDEFINED,3,VT_UI4,VT_UI4,VT_UI4)
    // typedef struct STAMPCOLLECTION
    // {
    //     DWORD LDevStamp;                LDev stamp
    //     DWORD AccoStampCDB;             ACCO stamp: persistent and volatile
    //                                     changes to CDB
    //     DWORD AccoStampModCounter;      ACCO stamp: modification counter
    // } STAMPCOLLECTION;
};

```

5.1.1.3.4 FAL ASE classes

5.1.1.3.4.1 CBAPhysicalDevice

```
[
    uuid(CBA00000-6C97-11D1-8271-00A02442DF7D),
    helpstring("Physical device class")
]
coclass CBAPhysicalDevice
{
    [default]    interface ICBAPhysicalDevice;
                 interface ICBAPhysicalDevice2;
                 interface ICBABrowse;
                 interface ICBABrowse2;
                 interface ICBAPersist;
                 interface ICBAPersist2;
};
```

5.1.1.3.4.2 CBALogicalDevice

```
[
    uuid(CBA00010-6C97-11D1-8271-00A02442DF7D),
    noncreatable,
    helpstring("Logical device class")
]
coclass CBALogicalDevice
{
    [default]    interface ICBALogicalDevice;
                 interface ICBALogicalDevice2;
                 interface ICBAState;
                 interface ICBAState2;
                 interface ICBABrowse;
                 interface ICBABrowse2;
                 interface ICBAGroupError;
                 interface ICBAPersist; // only used by PC implementation
};
```

5.1.1.3.4.3 CBAAcco

```
[
    uuid(CBA00040-6C97-11D1-8271-00A02442DF7D),
    noncreatable,
    helpstring("ACCO class")
]
coclass CBAAcco
{
    [default]    interface ICBAAccoMgt;
                 interface ICBAAccoMgt2;
                 interface ICBAAccoServer;
                 interface ICBAAccoServer2;
                 interface ICBAAccoServerSRT;
                 interface ICBAAccoCallback;
                 interface ICBAAccoCallback2;
                 interface ICBAAccoSync;
                 interface ICBAGroupError;
};
```

5.1.1.3.4.4 CBARTAuto

```
[
    uuid(CBA00050-6C97-11D1-8271-00A02442DF7D),
    noncreatable,
    helpstring("RTAuto class")
]
coclass CBARTAuto
{
    [default]    interface ICBARTAuto;
                 interface ICBARTAuto2;
                 interface ICBABrowse;
                 interface ICBABrowse2;
    // plus additional interface(s) for items
    // or ICBASystemProperties for special "!SYSTEM" RTAuto
};
```

5.1.1.3.4.5 CBASystemRTAuto

```
[
```

```
    uuid(CBA00090-6C97-11D1-8271-00A02442DF7D),  
    noncreatable,  
    helpstring("System RTAuto class")  
]  
coclass CBASystemRTAuto  
{  
    [default]    interface ICBARTAuto;  
                 interface ICBARTAuto2;  
                 interface ICBABrowse;  
                 interface ICBABrowse2;  
                 interface ICBASystemProperties;  
};
```

5.2 Transfer syntax

5.2.1 General

The transfer syntax is specified by the underlying ORPC wire protocol. The IDL defined by this specification shall be the basis for the APDU encoding and therefore the input for the ORPC model.

5.2.2 Serialization of values and parameter

5.2.2.1 General

Table 249 defines the serialization of the data types that are permitted for connections.

Table 249 – Data format for serialized connection data

Vartype	Data Type	Used Bytes in Data (ORPC channel)	Used Bytes in Data (RT channel)
VT_BOOL	2-byte signed int (True=-1, False=0)	2 bytes	
VT_I1	char	1 byte	
VT_UI1	unsigned char	1 byte	
VT_I2	short	2 bytes	
VT_UI2	unsigned short	2 bytes	
VT_I4	long	4 bytes	
VT_UI4	unsigned long	4 bytes	
VT_R4	float	4 bytes	
VT_R8	double	8 bytes	
VT_DATE	date, 8-byte real	8 bytes	
VT_BSTR	binary string	Byte count (VT_UI4) Afterwards (byte count / 2) * 2-byte wide chars	Byte count (VT_UI4) Afterwards (maximum byte count / 2) * 2-byte wide chars The maximum byte count of the consumer is used, even if the provider has a shorter BSTR, since the consumer assesses the layout of the buffer. Only the wide characters within the actual length of the BSTR are copied to the buffer, there is no need to null out the wide characters between actual length and maximum length; they are treated as don't care.
VT_SAFEARRAY	multidimensional array	Array dimension (VT_UI2) for each dimension number of elements (VT_UI4) Serialized elements of the basis data type according to their declaration. The array elements are (like in C/C++) serialized line by line. This means that the last index runs the fastest.	Array dimension and number of elements in each dimension is omitted. Serialized elements of the basis data type according to their declaration. The array elements (like in C/C++) are serialized line by line. This means that the last index runs the fastest.
VT_USERDEFINED	structure	The components of the structure are packed in the sequence of their declaration (e.g. without padding bytes) and stored as defined above.	

NOTE For an array the dimension and lengths will be omitted in the RT channel. The data type is clarified in the Connect service of the Acco Server SRT interface by transmitting the extended type description.

Arrays of BSTR and BSTR embedded in structures will always be transmitted with their maximum length; this applies to ORPC and RT channel.

5.2.2.2 Restrictions for the ORPC channel

Since a properties maximum connection data size is restricted to 32 kByte in the ORPC channel, only a maximum amount of data can be transmitted in a connection. This restricts the following data types, if they are used within a connection:

Strings can have up to 16 382 wide characters.

Arrays (one dimensional assumed) can have a maximum amount of data with 32 762 Byte.

Structures are limited to a maximum size of 32 768 Byte.

A device may have smaller maximum sizes of its properties.

5.2.2.3 Restrictions for the RT channel

Since the maximum length of an item in the format defined above is restricted to 450 bytes in the RT channel and consistency of values is not ensured across multiple AccoDataCRs, the following data types are restricted within their length, if they are used within a connection:

Strings can have up to 223 wide characters, since the actual byte count (4 bytes) is transmitted within the 450 bytes.

Arrays (one-dimensional assumed) can have a maximum amount of data with 450 byte.

Structures are limited to a maximum size of 450 byte.

NOTE Neither header nor item header are included in the maximum item size of 450 byte.

NOTE The maximum item size can have device specific further restrictions.

The size of the RT reference data is computed according to Table 250. It includes both overall header and all item headers. The maximum size of the RT reference data is 484 bytes, the minimum size is 40 bytes.

Table 250 – Calculation of the RT reference data size

Element	Needed length
Header	4 bytes
Per Item	For the Item Header 2 bytes
	For the Quality Code 1 byte
	For the data types VT_I1 and VT_UI1: 1 byte
	For the data types VT_BOOL, VT_I2 and VT_UI2: 2 bytes
	For the data types VT_I4, VT_UI4 and VT_R4: 4 bytes
	For the data type VT_R8: 8 bytes
	For the data type VT_BSTR: 4 bytes + maximum byte length
	For the data type VT_SAFEARRAY: $Len_1 * \dots * Len_{dim} * \text{space per item}$
	For the data type VT_USERDEFINED: Space per item

5.3 FAL protocol state machines

The FAL protocol state machine structure is as defined in Figure 50. The general structure is according to the type 1 protocol machine model.

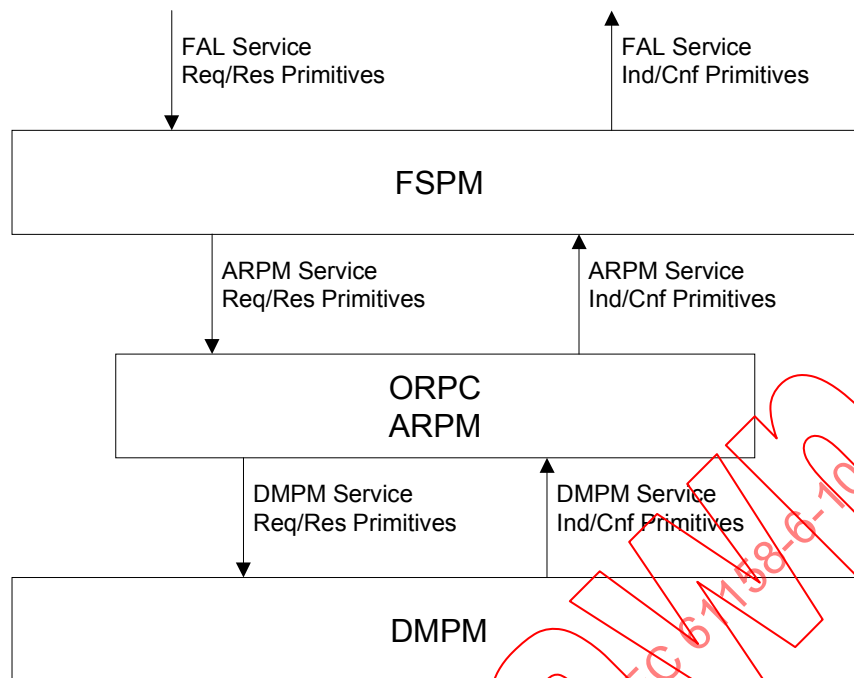


Figure 50 – Relationship among protocol machines

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. Currently, the type 10 specifies exactly one AREP class and shall have only one instance of this class. Therefore, the common service attribute AREP or AREP_ID shall be omitted and an implicit AREP selection is used. The addressing of the FAL object interface instance is specified via the common class and service attribute Interface Pointer.

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 ORPC ARPM state machine based on the implicit addressing mechanism and send FAL internal primitives with unified service parameters to the ORPC 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 or object instances addressed via the service attribute Interface Pointer.

The ARPM specifies the conveyance type for the application relation and adds the reference to the appropriated IDL to support the data marshaling of the underlying ORPC model.

The DMPM specifies the mapping to an abstract ORPC model. The ORPC model defines therefore an abstract service interface.

5.4 AP Context State Machine

There is no AP Context State Machine defined for the FAL AE.

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.

5.5.2 Primitive definitions

5.5.2.1 Primitives exchanged between FSPM and FAL User

Table 251 shows the service primitives including their associated parameters issued by the FAL User and received by the FSPM.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 251 – Primitives issued by FAL User to FSPM

Primitive name	Source	Associated parameters	Functions
QueryInterface.req	FAL User	Interface Pointer, Interface ID	Refer to FAL Service Definition in IEC 61158-5-10
QueryInterface.rsp(+)	FAL User	Interface Pointer, hresult, ppvObject	
QueryInterface.rsp(-)	FAL User	Interface Pointer, hresult	
AddRef.req	FAL User	Interface Pointer	
AddRef.rsp	FAL User	Interface Pointer, Reference Count	
Release.req	FAL User	Interface Pointer	
Release.rsp	FAL User	Interface Pointer, Reference Count	
GetTypeInfoCount.req	FAL User	Interface Pointer	
GetTypeInfoCount.rsp(+)	FAL User	Interface Pointer, hresult, pcTInfo	
GetTypeInfoCount.rsp(-)	FAL User	Interface Pointer, hresult	
GetTypeInfo.req	FAL User	Interface Pointer, iTInfo, lcId	
GetTypeInfo.rsp(+)	FAL User	Interface Pointer, hresult, ppTInfo	
GetTypeInfo.rsp(-)	FAL User	Interface Pointer, hresult	
GetIDsOfNames.req	FAL User	Interface Pointer, riid, List of rgpszNames, cNames, lcId	
GetIDsOfNames.rsp(+)	FAL User	Interface Pointer, hresult, List of rgDispId	
GetIDsOfNames.rsp(-)	FAL User	Interface Pointer, hresult	
Invoke.req	FAL User	Interface Pointer, dispIdMember, riid, lcId, wFlags, pDispParams	
Invoke.rsp(+)	FAL User	Interface Pointer, hresult, pVarResult	
Invoke.rsp(-)	FAL User	Interface Pointer, hresult, pExcepInfo, puArgErr	
get_Producer.req	FAL User	Interface Pointer	
get_Producer.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Producer.rsp(-)	FAL User	Interface Pointer, hresult	
get_Product.req	FAL User	Interface Pointer	
get_Product.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Product.rsp(-)	FAL User	Interface Pointer, hresult	
get_SerialNo.req	FAL User	Interface Pointer	
get_SerialNo.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_SerialNo.rsp(-)	FAL User	Interface Pointer, hresult	
get_ProductionDate.req	FAL User	Interface Pointer	
get_ProductionDate.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_ProductionDate.rsp(-)	FAL User	Interface Pointer, hresult	
Revision.req	FAL User	Interface Pointer	
Revision.rsp(+)	FAL User	Interface Pointer, hresult, pMajor, pMinor	

Primitive name	Source	Associated parameters	Functions
Revision.rsp(-)	FAL User	Interface Pointer, hresult	
get_LogicalDevice.req	FAL User	Interface Pointer, Name	
get_LogicalDevice.rsp(+)	FAL User	Interface Pointer, hresult, ppLDev	
get_LogicalDevice.rsp(-)	FAL User	Interface Pointer, hresult	
Type.req	FAL User	Interface Pointer	
Type.rsp(+)	FAL User	Interface Pointer, hresult, pMultiApp, pPROFInetDcomStack	
Type.rsp(-)	FAL User	Interface Pointer, hresult	
PROFInetRevision.req	FAL User	Interface Pointer	
PROFInetRevision.rsp(+)	FAL User	Interface Pointer, hresult, pMajor, pMinor, pServicePack, pBuild	
PROFInetRevision.rsp(-)	FAL User	Interface Pointer, hresult	
get_PDevStamp.req	FAL User	Interface Pointer	
get_PDevStamp.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_PDevStamp.rsp(-)	FAL User	Interface Pointer, hresult	
get_Count.req	FAL User	Interface Pointer	
get_Count.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Count.rsp(-)	FAL User	Interface Pointer, hresult	
BrowseItems.req	FAL User	Interface Pointer, Offset, MaxReturn	
BrowseItems.rsp(+)	FAL User	Interface Pointer, hresult, pItem, pDataType, pAccessRight	
BrowseItems.rsp(-)	FAL User	Interface Pointer, hresult	
get_Count2.req	FAL User	Interface Pointer, Selector	
get_Count2.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Count2.rsp(-)	FAL User	Interface Pointer, hresult	
BrowseItems2.req	FAL User	Interface Pointer, Selector, Offset, MaxReturn	
BrowseItems2.rsp(+)	FAL User	Interface Pointer, hresult, pItem, pInfo1, pInfo2	
BrowseItems2.rsp(-)	FAL User	Interface Pointer, hresult	
Save.req	FAL User	Interface Pointer	
Save.rsp(+)	FAL User	Interface Pointer, hresult	
Save.rsp(-)	FAL User	Interface Pointer, hresult	
Save2.req	FAL User	Interface Pointer	
Save2.rsp(+)	FAL User	Interface Pointer, hresult, pLDevName, pPartialResult	
Save2.rsp(-)	FAL User	Interface Pointer, hresult	
get_Name.req	FAL User	Interface Pointer	
get_Name.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Name.rsp(-)	FAL User	Interface Pointer, hresult	
get_ACCO.req	FAL User	Interface Pointer	

Primitive name	Source	Associated parameters	Functions
get_ACCO.rsp(+)	FAL User	Interface Pointer, hresult, ppACCO	
get_ACCO.rsp(-)	FAL User	Interface Pointer, hresult	
get_RTAuto.req	FAL User	Interface Pointer, Name	
get_RTAuto.rsp(+)	FAL User	Interface Pointer, hresult, ppAuto	
get_RTAuto.rsp(-)	FAL User	Interface Pointer, hresult	
ComponentInfo.req	FAL User	Interface Pointer	
ComponentInfo.rsp(+)	FAL User	Interface Pointer, hresult, pComponentID, pVersion	
ComponentInfo.rsp(-)	FAL User	Interface Pointer, hresult	
get_State.req	FAL User	Interface Pointer	
get_State.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_State.rsp(-)	FAL User	Interface Pointer, hresult	
Activate.req	FAL User	Interface Pointer	
Activate.rsp(+)	FAL User	Interface Pointer, hresult	
Activate.rsp(-)	FAL User	Interface Pointer, hresult	
Deactivate.req	FAL User	Interface Pointer	
Deactivate.rsp(+)	FAL User	Interface Pointer, hresult	
Deactivate.rsp(-)	FAL User	Interface Pointer, hresult	
Reset.req	FAL User	Interface Pointer	
Reset.rsp(+)	FAL User	Interface Pointer, hresult	
Reset.rsp(-)	FAL User	Interface Pointer, hresult	
AdviseState.req	FAL User	Interface Pointer, pStateEvent	
AdviseState.rsp(+)	FAL User	Interface Pointer, hresult, pCookie	
AdviseState.rsp(-)	FAL User	Interface Pointer, hresult	
UnadviseState.req	FAL User	Interface Pointer, Cookie	
UnadviseState.rsp(+)	FAL User	Interface Pointer, hresult	
UnadviseState.rsp(-)	FAL User	Interface Pointer, hresult	
get_Time.req	FAL User	Interface Pointer	
get_Time.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Time.rsp(-)	FAL User	Interface Pointer, hresult	
put_Time.req	FAL User	Interface Pointer, NewVal	
put_Time.rsp(+)	FAL User	Interface Pointer, hresult	
put_Time.rsp(-)	FAL User	Interface Pointer, hresult	
GroupError.req	FAL User	Interface Pointer	
GroupError.rsp(+)	FAL User	Interface Pointer, hresult, pVal, pMagicCookie	
GroupError.rsp(-)	FAL User	Interface Pointer, hresult	
AdviseGroupError.req	FAL User	Interface Pointer, pGroupErrorEvent	
AdviseGroupError.rsp(+)	FAL User	Interface Pointer, hresult, pCookie	
AdviseGroupError.rsp(-)	FAL User	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
UnAdviseGroupError.req	FAL User	Interface Pointer, Cookie	
UnAdviseGroupError.rsp(+)	FAL User	Interface Pointer, hresult	
UnAdviseGroupError.rsp(-)	FAL User	Interface Pointer, hresult	
AddConnections.req	FAL User	Interface Pointer, Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn	
AddConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppAddConnectionOut	
AddConnections.rsp(-)	FAL User	Interface Pointer, hresult	
RemoveConnections.req	FAL User	Interface Pointer, Count, List of pConsumerID	
RemoveConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
RemoveConnections.rsp(-)	FAL User	Interface Pointer, hresult	
ClearConnections.req	FAL User	Interface Pointer	
ClearConnections.rsp(+)	FAL User	Interface Pointer, hresult	
ClearConnections.rsp(-)	FAL User	Interface Pointer, hresult	
SetActivationState.req	FAL User	Interface Pointer, State, Count, List of pConsumerID	
SetActivationState.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
SetActivationState.rsp(-)	FAL User	Interface Pointer, hresult	
GetInfo.req	FAL User	Interface Pointer	
GetInfo.rsp(+)	FAL User	Interface Pointer, hresult, pMax, pCurCnt	
GetInfo.rsp(-)	FAL User	Interface Pointer, hresult	
GetIDs.req	FAL User	Interface Pointer	
GetIDs.rsp(+)	FAL User	Interface Pointer, hresult, pCount, List of ppGetIDOut	
GetIDs.rsp(-)	FAL User	Interface Pointer, hresult	
GetConnections.req	FAL User	Interface Pointer, Count, List of pConsumerID	
GetConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppGetConnectionOut	
GetConnections.rsp(-)	FAL User	Interface Pointer, hresult	
ReviseQoS.req	FAL User	Interface Pointer, RTAuto, QoSType, QoSValue	
ReviseQoS.rsp(+)	FAL User	Interface Pointer, hresult, pRevisedQoSValue	
ReviseQoS.rsp(-)	FAL User	Interface Pointer, hresult	
get_PingFactor.req	FAL User	Interface Pointer	
get_PingFactor.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_PingFactor.rsp(-)	FAL User	Interface Pointer, hresult	
put_PingFactor.req	FAL User	Interface Pointer, NewVal	
put_PingFactor.rsp(+)	FAL User	Interface Pointer, hresult	
put_PingFactor.rsp(-)	FAL User	Interface Pointer, hresult	
get_CDBCcookie.req	FAL User	Interface Pointer	
get_CDBCcookie.rsp(+)	FAL User	Interface Pointer, hresult, pVal	

Primitive name	Source	Associated parameters	Functions
get_CDBCcookie.rsp(-)	FAL User	Interface Pointer, hresult	
GetConsIDs.req	FAL User	Interface Pointer	
GetConsIDs.rsp(+)	FAL User	Interface Pointer, hresult, pCount, List of ppConsumerID	
GetConsIDs.rsp(-)	FAL User	Interface Pointer, hresult	
GetConsConnections.req	FAL User	Interface Pointer, Count, List of pConsumerID	
GetConsConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppGetConsConnOut	
GetConsConnections.rsp(-)	FAL User	Interface Pointer, hresult	
DiagConsConnections.req	FAL User	Interface Pointer, Count, List of pConsumerID	
DiagConsConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppDiagConsConnOut	
DiagConsConnections.rsp(-)	FAL User	Interface Pointer, hresult	
GetProvIDs.req	FAL User	Interface Pointer	
GetProvIDs.rsp(+)	FAL User	Interface Pointer, hresult, pCount, List of ppProviderID	
GetProvIDs.rsp(-)	FAL User	Interface Pointer, hresult	
GetProvConnections.req	FAL User	Interface Pointer, Count, List of pProviderID	
GetProvConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppGetProvConnOut	
GetProvConnections.rsp(-)	FAL User	Interface Pointer, hresult	
GetDiagnosis.req	FAL User	Interface Pointer, Request, InLength, pInBuffer	
GetDiagnosis.rsp(+)	FAL User	Interface Pointer, hresult, pOutLength, ppOutBuffer	
GetDiagnosis.rsp(-)	FAL User	Interface Pointer, hresult	
Connect.req	FAL User	Interface Pointer, Consumer, QoSType, QoSValue, State, pCBAAccoCallback, Count, List of pConnectIn	
Connect.rsp(+)	FAL User	Interface Pointer, hresult, pFirstConnect, List of ppConnectOut	
Connect.rsp(-)	FAL User	Interface Pointer, hresult	
Disconnect.req	FAL User	Interface Pointer, Count, List of pProviderID	
Disconnect.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
Disconnect.rsp(-)	FAL User	Interface Pointer, hresult	
DisconnectMe.req	FAL User	Interface Pointer, Consumer	
DisconnectMe.rsp(+)	FAL User	Interface Pointer, hresult	
DisconnectMe.rsp(-)	FAL User	Interface Pointer, hresult	
SetActivation.req	FAL User	Interface Pointer, State, Count, List of pProviderID	
SetActivation.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
SetActivation.rsp(-)	FAL User	Interface Pointer, hresult	
Ping.req	FAL User	Interface Pointer, Consumer	

Primitive name	Source	Associated parameters	Functions
Ping.rsp(+)	FAL User	Interface Pointer, hresult	
Ping.rsp(-)	FAL User	Interface Pointer, hresult	
Connect2.req	FAL User	Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn	
Connect2.rsp(+)	FAL User	Interface Pointer, hresult, pFirstConnect, List of ppConnectOut	
Connect2.rsp(-)	FAL User	Interface Pointer, hresult	
GetConnectionData.req	FAL User	Interface Pointer, Consumer	
GetConnectionData.rsp(+)	FAL User	Interface Pointer, hresult, pLength, pBuffer	
GetConnectionData.rsp(-)	FAL User	Interface Pointer, hresult	
OnDataChanged.req	FAL User	Interface Pointer, Length, pBuffer	
OnDataChanged.rsp(+)	FAL User	Interface Pointer, hresult	
OnDataChanged.rsp(-)	FAL User	Interface Pointer, hresult	
Gnip.req	FAL User	Interface Pointer	
Gnip.rsp(+)	FAL User	Interface Pointer, hresult	
Gnip.rsp(-)	FAL User	Interface Pointer, hresult	
ReadItems.req	FAL User	Interface Pointer, Count, List of pReadItem	
ReadItems.rsp(+)	FAL User	Interface Pointer, hresult, List of ppReadItemOut	
ReadItems.rsp(-)	FAL User	Interface Pointer, hresult	
WriteItems.req	FAL User	Interface Pointer, Count, List of pWriteItem	
WriteItems.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
WriteItems.rsp(-)	FAL User	Interface Pointer, hresult	
WriteItemsQCD.req	FAL User	Interface Pointer, Count, List of pWriteItemQcdIn	
WriteItemsQCD.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
WriteItemsQCD.rsp(-)	FAL User	Interface Pointer, hresult	
ConnectCR.req	FAL User	Interface Pointer, Consumer, QoSType, QoSValue, pICBAAccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn	
ConnectCR.rsp(+)	FAL User	Interface Pointer, hresult, pFirstConnect, pProviderMAC, List of ppConnectOut	
ConnectCR.rsp(-)	FAL User	Interface Pointer, hresult	
DisconnectCR.req	FAL User	Interface Pointer, Count, List of pProviderCRID	
DisconnectCR.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
DisconnectCR.rsp(-)	FAL User	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
SRTConnect.req	FAL User	Interface Pointer, ProviderCRID, State, LastConnect, Count, List of pConnectIn	
SRTConnect.rsp(+)	FAL User	Interface Pointer, hresult, List of ppConnectOut	
SRTConnect.rsp(-)	FAL User	Interface Pointer, hresult	
CoCreateInstance.req	FAL User	Host address, Class ID, Interface ID,	
CoCreateInstance.rsp(+)	FAL User	hresult, Interface Pointer	
CoCreateInstance.rsp(-)	FAL User	hresult	
CoDisconnectObject.req	FAL User	Interface Pointer	
CoDisconnectObject.rsp	FAL User	Interface Pointer, hresult	
Call.req	FSPM	Interface Pointer, Service Name, List of Unified Service In-Parameter	
Call.rsp(+)	FSPM	Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter	
Call.rsp(-)	FSPM	Interface Pointer, Service Name, hresult	

Table 252 shows the service primitives including their associated parameters issued by the FSPM and received by the FAL User.

Table 252 – Primitives issued by FSPM to FAL User

Primitive name	Source	Associated parameters	Functions
QueryInterface.ind	FSPM	Interface Pointer, Interface ID	Refer to FAL Service Definition in IEC 61158-5-10
QueryInterface.cnf(+)	FSPM	Interface Pointer, hresult, ppvObject	
QueryInterface.cnf(-)	FSPM	Interface Pointer, hresult	
AddRef.ind	FSPM	Interface Pointer	
AddRef.cnf	FSPM	Interface Pointer, Reference Count	
Release.ind	FSPM	Interface Pointer	
Release.cnf	FSPM	Interface Pointer, Reference Count	
GetTypeInfoCount.ind	FSPM	Interface Pointer	
GetTypeInfoCount.cnf(+)	FSPM	Interface Pointer, hresult, pcTInfo	
GetTypeInfoCount.cnf(-)	FSPM	Interface Pointer, hresult	
GetTypeInfo.ind	FSPM	Interface Pointer, iTInfo, lcId	
GetTypeInfo.cnf(+)	FSPM	Interface Pointer, hresult, ppTInfo	
GetTypeInfo.cnf(-)	FSPM	Interface Pointer, hresult	
GetIDsOfNames.ind	FSPM	Interface Pointer, riid, List of rgpszNames, cNames, lcId	
GetIDsOfNames.cnf(+)	FSPM	Interface Pointer, hresult, List of rgDispId	
GetIDsOfNames.cnf(-)	FSPM	Interface Pointer, hresult	
Invoke.ind	FSPM	Interface Pointer, dispIdMember, riid, lcId, wFlags, pDispParams	
Invoke.cnf(+)	FSPM	Interface Pointer, hresult, pVarResult	
Invoke.cnf(-)	FSPM	Interface Pointer, hresult, pExcepInfo, puArgErr	
get_Producer.ind	FSPM	Interface Pointer	
get_Producer.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Producer.cnf(-)	FSPM	Interface Pointer, hresult	
get_Product.ind	FSPM	Interface Pointer	
get_Product.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Product.cnf(-)	FSPM	Interface Pointer, hresult	
get_SerialNo.ind	FSPM	Interface Pointer	
get_SerialNo.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_SerialNo.cnf(-)	FSPM	Interface Pointer, hresult	
get_ProductionDate.ind	FSPM	Interface Pointer	
get_ProductionDate.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_ProductionDate.cnf(-)	FSPM	Interface Pointer, hresult	
Revision.ind	FSPM	Interface Pointer	
Revision.cnf(+)	FSPM	Interface Pointer, hresult, pMajor, pMinor	

Primitive name	Source	Associated parameters	Functions
Revision.cnf(-)	FSPM	Interface Pointer, hresult	
get_LogicalDevice.ind	FSPM	Interface Pointer, Name	
get_LogicalDevice.cnf(+)	FSPM	Interface Pointer, hresult, ppLDev	
get_LogicalDevice.cnf(-)	FSPM	Interface Pointer, hresult	
Type.ind	FSPM	Interface Pointer	
Type.cnf(+)	FSPM	Interface Pointer, hresult, pMultiApp, pPROFInetDcomStack	
Type.cnf(-)	FSPM	Interface Pointer, hresult	
PROFInetRevision.ind	FSPM	Interface Pointer	
PROFInetRevision.cnf(+)	FSPM	Interface Pointer, hresult, pMajor, pMinor, pServicePack, pBuild	
PROFInetRevision.cnf(-)	FSPM	Interface Pointer, hresult	
get_PDevStamp.ind	FSPM	Interface Pointer	
get_PDevStamp.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_PDevStamp.cnf(-)	FSPM	Interface Pointer, hresult	
get_Count.ind	FSPM	Interface Pointer	
get_Count.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Count.cnf(-)	FSPM	Interface Pointer, hresult	
BrowseItems.ind	FSPM	Interface Pointer, Offset, MaxReturn	
BrowseItems.cnf(+)	FSPM	Interface Pointer, hresult, pItem, pDataType, pAccessRight	
BrowseItems.cnf(-)	FSPM	Interface Pointer, hresult	
get_Count2.ind	FSPM	Interface Pointer, Selector	
get_Count2.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Count2.cnf(-)	FSPM	Interface Pointer, hresult	
BrowseItems2.ind	FSPM	Interface Pointer, Selector, Offset, MaxReturn	
BrowseItems2.cnf(+)	FSPM	Interface Pointer, hresult, pItem, pInfo1, pInfo2	
BrowseItems2.cnf(-)	FSPM	Interface Pointer, hresult	
Save.ind	FSPM	Interface Pointer	
Save.cnf(+)	FSPM	Interface Pointer, hresult	
Save.cnf(-)	FSPM	Interface Pointer, hresult	
Save2.ind	FSPM	Interface Pointer	
Save2.cnf(+)	FSPM	Interface Pointer, hresult, pLDevName, pPartialResult	
Save2.cnf(-)	FSPM	Interface Pointer, hresult	
get_Name.ind	FSPM	Interface Pointer	
get_Name.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Name.cnf(-)	FSPM	Interface Pointer, hresult	
get_ACCO.ind	FSPM	Interface Pointer	

Primitive name	Source	Associated parameters	Functions
get_ACCO.cnf(+)	FSPM	Interface Pointer, hresult, ppACCO	
get_ACCO.cnf(-)	FSPM	Interface Pointer, hresult	
get_RTAuto.ind	FSPM	Interface Pointer, Name	
get_RTAuto.cnf(+)	FSPM	Interface Pointer, hresult, ppAuto	
get_RTAuto.cnf(-)	FSPM	Interface Pointer, hresult	
ComponentInfo.ind	FSPM	Interface Pointer	
ComponentInfo.cnf(+)	FSPM	Interface Pointer, hresult, pComponentID, pVersion	
ComponentInfo.cnf(-)	FSPM	Interface Pointer, hresult	
get_State.ind	FSPM	Interface Pointer	
get_State.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_State.cnf(-)	FSPM	Interface Pointer, hresult	
Activate.ind	FSPM	Interface Pointer	
Activate.cnf(+)	FSPM	Interface Pointer, hresult	
Activate.cnf(-)	FSPM	Interface Pointer, hresult	
Deactivate.ind	FSPM	Interface Pointer	
Deactivate.cnf(+)	FSPM	Interface Pointer, hresult	
Deactivate.cnf(-)	FSPM	Interface Pointer, hresult	
Reset.ind	FSPM	Interface Pointer	
Reset.cnf(+)	FSPM	Interface Pointer, hresult	
Reset.cnf(-)	FSPM	Interface Pointer, hresult	
AdviseState.ind	FSPM	Interface Pointer, pStateEvent	
AdviseState.cnf(+)	FSPM	Interface Pointer, hresult, pCookie	
AdviseState.cnf(-)	FSPM	Interface Pointer, hresult	
UnadviseState.ind	FSPM	Interface Pointer, Cookie	
UnadviseState.cnf(+)	FSPM	Interface Pointer, hresult	
UnadviseState.cnf(-)	FSPM	Interface Pointer, hresult	
get_Time.ind	FSPM	Interface Pointer	
get_Time.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Time.cnf(-)	FSPM	Interface Pointer, hresult	
put_Time.ind	FSPM	Interface Pointer, NewVal	
put_Time.cnf(+)	FSPM	Interface Pointer, hresult	
put_Time.cnf(-)	FSPM	Interface Pointer, hresult	
GroupError.ind	FSPM	Interface Pointer	
GroupError.cnf(+)	FSPM	Interface Pointer, hresult, pVal, pMagicCookie	
GroupError.cnf(-)	FSPM	Interface Pointer, hresult	
AdviseGroupError.ind	FSPM	Interface Pointer, pGroupErrorEvent	
AdviseGroupError.cnf(+)	FSPM	Interface Pointer, hresult, pCookie	
AdviseGroupError.cnf(-)	FSPM	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
UnAdviseGroupError.ind	FSPM	Interface Pointer, Cookie	
UnAdviseGroupError.cnf(+)	FSPM	Interface Pointer, hresult	
UnAdviseGroupError.cnf(-)	FSPM	Interface Pointer, hresult	
AddConnections.ind	FSPM	Interface Pointer, Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn	
AddConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppAddConnectionOut	
AddConnections.cnf(-)	FSPM	Interface Pointer, hresult	
RemoveConnections.ind	FSPM	Interface Pointer, Count, List of pConsumerID	
RemoveConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
RemoveConnections.cnf(-)	FSPM	Interface Pointer, hresult	
ClearConnections.ind	FSPM	Interface Pointer	
ClearConnections.cnf(+)	FSPM	Interface Pointer, hresult	
ClearConnections.cnf(-)	FSPM	Interface Pointer, hresult	
SetActivationState.ind	FSPM	Interface Pointer, State, Count, List of pConsumerID	
SetActivationState.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
SetActivationState.cnf(-)	FSPM	Interface Pointer, hresult	
GetInfo.ind	FSPM	Interface Pointer	
GetInfo.cnf(+)	FSPM	Interface Pointer, hresult, pMax, pCurCnt	
GetInfo.cnf(-)	FSPM	Interface Pointer, hresult	
GetIDs.ind	FSPM	Interface Pointer	
GetIDs.cnf(+)	FSPM	Interface Pointer, hresult, pCount, List of ppGetIDOut	
GetIDs.cnf(-)	FSPM	Interface Pointer, hresult	
GetConnections.ind	FSPM	Interface Pointer, Count, List of pConsumerID	
GetConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppGetConnectionOut	
GetConnections.cnf(-)	FSPM	Interface Pointer, hresult	
ReviseQoS.ind	FSPM	Interface Pointer, RTAuto, QoSType, QoSValue	
ReviseQoS.cnf(+)	FSPM	Interface Pointer, hresult, pRevisedQoSValue	
ReviseQoS.cnf(-)	FSPM	Interface Pointer, hresult	
get_PingFactor.ind	FSPM	Interface Pointer	
get_PingFactor.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_PingFactor.cnf(-)	FSPM	Interface Pointer, hresult	
put_PingFactor.ind	FSPM	Interface Pointer, NewVal	
put_PingFactor.cnf(+)	FSPM	Interface Pointer, hresult	
put_PingFactor.cnf(-)	FSPM	Interface Pointer, hresult	
get_CDBCcookie.ind	FSPM	Interface Pointer	
get_CDBCcookie.cnf(+)	FSPM	Interface Pointer, hresult, pVal	

Primitive name	Source	Associated parameters	Functions
get_CDBCcookie.cnf(-)	FSPM	Interface Pointer, hresult	
GetConsIDs.ind	FSPM	Interface Pointer	
GetConsIDs.cnf(+)	FSPM	Interface Pointer, hresult, pCount, List of ppConsumerID	
GetConsIDs.cnf(-)	FSPM	Interface Pointer, hresult	
GetConsConnections.ind	FSPM	Interface Pointer, Count, List of pConsumerID	
GetConsConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppGetConsConnOut	
GetConsConnections.cnf(-)	FSPM	Interface Pointer, hresult	
DiagConsConnections.ind	FSPM	Interface Pointer, Count, List of pConsumerID	
DiagConsConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppDiagConsConnOut	
DiagConsConnections.cnf(-)	FSPM	Interface Pointer, hresult	
GetProvIDs.ind	FSPM	Interface Pointer	
GetProvIDs.cnf(+)	FSPM	Interface Pointer, hresult, pCount, List of ppProviderID	
GetProvIDs.cnf(-)	FSPM	Interface Pointer, hresult	
GetProvConnections.ind	FSPM	Interface Pointer, Count, List of pProviderID	
GetProvConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppGetProvConnOut	
GetProvConnections.cnf(-)	FSPM	Interface Pointer, hresult	
GetDiagnosis.ind	FSPM	Interface Pointer, Request, InLength, pInBuffer	
GetDiagnosis.cnf(+)	FSPM	Interface Pointer, hresult, pOutLength, ppOutBuffer	
GetDiagnosis.cnf(-)	FSPM	Interface Pointer, hresult	
Connect.ind	FSPM	Interface Pointer, Consumer, QoSType, QoSValue, State, pCBAAccoCallback, Count, List of pConnectIn	
Connect.cnf(+)	FSPM	Interface Pointer, hresult, pFirstConnect, List of ppConnectOut	
Connect.cnf(-)	FSPM	Interface Pointer, hresult	
Disconnect.ind	FSPM	Interface Pointer, Count, List of pProviderID	
Disconnect.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
Disconnect.cnf(-)	FSPM	Interface Pointer, hresult	
DisconnectMe.ind	FSPM	Interface Pointer, Consumer	
DisconnectMe.cnf(+)	FSPM	Interface Pointer, hresult	
DisconnectMe.cnf(-)	FSPM	Interface Pointer, hresult	
SetActivation.ind	FSPM	Interface Pointer, State, Count, List of pProviderID	
SetActivation.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
SetActivation.cnf(-)	FSPM	Interface Pointer, hresult	
Ping.ind	FSPM	Interface Pointer, Consumer	

Primitive name	Source	Associated parameters	Functions
Ping.cnf(+)	FSPM	Interface Pointer, hresult	
Ping.cnf(-)	FSPM	Interface Pointer, hresult	
Connect2.ind	FSPM	Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAaccoCallback, Count, List of pConnectIn	
Connect2.cnf(+)	FSPM	Interface Pointer, hresult, pFirstConnect, List of ppConnectOut	
Connect2.cnf(-)	FSPM	Interface Pointer, hresult	
OnDataChanged.ind	FSPM	Interface Pointer, Length, pBuffer	
GetConnectionData.ind	FSPM	Interface Pointer, Consumer	
GetConnectionData.cnf(+)	FSPM	Interface Pointer, hresult, pLength, pBuffer	
GetConnectionData.cnf(-)	FSPM	Interface Pointer, hresult	
OnDataChanged.cnf(+)	FSPM	Interface Pointer, hresult	
OnDataChanged.cnf(-)	FSPM	Interface Pointer, hresult	
Gnip.ind	FSPM	Interface Pointer	
Gnip.cnf(+)	FSPM	Interface Pointer, hresult	
Gnip.cnf(-)	FSPM	Interface Pointer, hresult	
ReadItems.ind	FSPM	Interface Pointer, Count, List of pReadItem	
ReadItems.cnf(+)	FSPM	Interface Pointer, hresult, List of ppReadItemOut	
ReadItems.cnf(-)	FSPM	Interface Pointer, hresult	
WriteItems.ind	FSPM	Interface Pointer, Count, List of pWriteItem	
WriteItems.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
WriteItems.cnf(-)	FSPM	Interface Pointer, hresult	
WriteItemsQCD.ind	FSPM	Interface Pointer, Count, List of pWriteItemQcdIn	
WriteItemsQCD.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
WriteItemsQCD.cnf(-)	FSPM	Interface Pointer, hresult	
ConnectCR.ind	FSPM	Interface Pointer, Consumer, QoSType, QoSValue, pICBAaccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn	
ConnectCR.cnf(+)	FSPM	Interface Pointer, hresult, pFirstConnect, pProviderMAC, List of ppConnectOut	
ConnectCR.cnf(-)	FSPM	Interface Pointer, hresult	
DisconnectCR.ind	FSPM	Interface Pointer, Count, List of pProviderCRID	
DisconnectCR.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
DisconnectCR.cnf(-)	FSPM	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
SRTConnect.ind	FSPM	Interface Pointer, ProviderCRID, State, LastConnect, Count, List of pConnectIn	
SRTConnect.cnf(+)	FSPM	Interface Pointer, hresult, List of ppConnectOut	
SRTConnect.cnf(-)	FSPM	Interface Pointer, hresult	
CoCreateInstance.ind	FSPM	Host address, Class ID, Interface ID,	
CoCreateInstance.cnf(+)	FSPM	hresult, Interface Pointer	
CoCreateInstance.cnf(-)	FSPM	hresult	
CoDisconnectObject.ind	FSPM	Interface Pointer	
CoDisconnectObject.cnf	FSPM	Interface Pointer, hresult	
Call.ind	FSPM	Interface Pointer, Service Name, List of Unified Service In-Parameter	
Call.cnf(+)	FSPM	Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter	
Call.cnf(-)	FSPM	Interface Pointer, Service Name, hresult	

5.5.2.2 Parameters of FAL User/FSPM Primitives

The parameters used with the primitives exchanged between the FAL User and the FSPM are described in FAL Service Definition in IEC 61158-5-10.

5.5.2.3 FSPM states

The defined state of the FSPM together with the description is listed in Table 253.

Table 253 – FSPM state descriptions

State Name	Description
ACTIVE	The FSPM in the ACTIVE state is ready to transmit and receive service primitives to and from the ARPM and the FAL user.

The state transition diagram of the FSPM is shown in Figure 51.

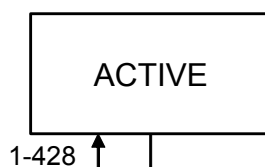


Figure 51 – State transition diagram of FSPM

5.5.2.4 FSPM state table

The FSPM state transitions are specified in Table 254.

Table 254 – FSPM state table

#	Current State	Event /Condition =>Action	Next State
1	ACTIVE	QueryInterface.req(Interface Pointer, Interface ID) => Interface Pointer:= Interface Pointer Service Name:=QueryInterface List of Unified Service In-Parameter:=Interface ID ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
2	ACTIVE	QueryInterface.rsp(+)(Interface Pointer, hresult, ppvObject) => Interface Pointer:= Interface Pointer Service Name:=QueryInterface hresult:=hresult List of Unified Service Out-Parameter:=ppvObject ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
3	ACTIVE	QueryInterface.rsp(-)(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer Service Name:=QueryInterface hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
4	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=QueryInterface => Interface Pointer:=Interface Pointer Interface ID:=List of Unified Service In-Parameter QueryInterface.ind(Interface Pointer, Interface ID)	ACTIVE
5	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=QueryInterface && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppvObject:=List of Unified Service Out-Parameter QueryInterface.cnf(+)(Interface Pointer, hresult, ppvObject)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
6	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=QueryInterface && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult QueryInterface.cnf(-)(Interface Pointer, hresult)	ACTIVE
7	ACTIVE	AddRef.req(Interface Pointer) => Interface Pointer:= Interface Pointer Service Name:=AddRef List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
8	ACTIVE	AddRef.rsp(Interface Pointer, Reference Count) => Interface Pointer:= Interface Pointer Service Name:=AddRef hresult:="empty" List of Unified Service Out-Parameter:=Reference Count ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
9	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=AddRef => Interface Pointer:=Interface Pointer AddRef.ind(Interface Pointer)	ACTIVE
10	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=AddRef => Interface Pointer:=Interface Pointer Reference Count:=List of Unified Service Out-Parameter AddRef.cnf(Interface Pointer, Reference Count)	ACTIVE
11	ACTIVE	Release.req(Interface Pointer) => Interface Pointer:= Interface Pointer Service Name:=Release List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
12	ACTIVE	Release.rsp(Interface Pointer, Reference Count) => Interface Pointer:= Interface Pointer Service Name:=Release hresult:="empty" List of Unified Service Out-Parameter:=Reference Count ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
13	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Release => Interface Pointer:=Interface Pointer Release.ind(Interface Pointer)	ACTIVE
14	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Release => Interface Pointer:=Interface Pointer Reference Count:=List of Unified Service Out-Parameter Release.cnf(Interface Pointer, Reference Count)	ACTIVE
15	ACTIVE	GetTypeInfoCount.req(Interface Pointer) => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfoCount List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
16	ACTIVE	GetTypeInfoCount.rsp(+)(Interface Pointer, hresult, pcTInfo) => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfoCount hresult:=hresult List of Unified Service Out-Parameter:=pcTInfo ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
17	ACTIVE	GetTypeInfoCount.rsp(-)(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfoCount hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
18	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetTypeInfoCount => Interface Pointer:=Interface Pointer GetTypeInfoCount.ind(Interface Pointer)	ACTIVE
19	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetTypeInfoCount && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pcTInfo:=List of Unified Service Out-Parameter GetTypeInfoCount.cnf(+)(Interface Pointer, hresult, pcTInfo)	ACTIVE
20	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetTypeInfoCount && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetTypeInfoCount.cnf(-)(Interface Pointer, hresult)	ACTIVE
21	ACTIVE	GetTypeInfo.req(Interface Pointer, iTInfo, Icid) => Interface Pointer:=Interface Pointer Service Name:=GetTypeInfo List of Unified Service In-Parameter:=(iTInfo, Icid) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
22	ACTIVE	GetTypeInfo.rsp(+)(Interface Pointer, hresult, ppTInfo) => Interface Pointer:=Interface Pointer Service Name:=GetTypeInfo hresult:=hresult List of Unified Service Out-Parameter:=ppTInfo ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
23	ACTIVE	GetTypeInfo.rsp(-)(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
24	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetTypeInfo => Interface Pointer:=Interface Pointer Icid:=List of Unified Service In-Parameter(Icid) iTypeInfo:=List of Unified Service In-Parameter(iTypeInfo) GetTypeInfo.ind(Interface Pointer, iTypeInfo, Icid)	ACTIVE
25	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetTypeInfo && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppTypeInfo:=List of Unified Service Out-Parameter GetTypeInfo.cnf(+)(Interface Pointer, hresult, ppTypeInfo)	ACTIVE
26	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetTypeInfo && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetTypeInfo.cnf(-)(Interface Pointer, hresult)	ACTIVE
27	ACTIVE	GetIDsOfNames.req(Interface Pointer, riid, List of rgpszNames, cNames, Icid) => Interface Pointer:= Interface Pointer Service Name:=GetIDsOfNames List of Unified Service In-Parameter:=(riid, List of rgpszNames, cNames, Icid) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
28	ACTIVE	GetIDsOfNames.rsp(+)(Interface Pointer, hresult, List of rgDispld) => Interface Pointer:= Interface Pointer Service Name:=GetIDsOfNames hresult:=hresult List of Unified Service Out-Parameter:=List of rgDispld ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
29	ACTIVE	GetIDsOfNames.rsp(-)(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer Service Name:=GetIDsOfNames hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
30	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetIDsOfNames => Interface Pointer:=Interface Pointer riid:=List of Unified Service In-Parameter(riid) List of rgpszNames:=List of Unified Service In-Parameter(List of rgpszNames) cNames:=List of Unified Service In-Parameter(cNames) Icid:=List of Unified Service In-Parameter(Icid) GetIDsOfNames.ind(Interface Pointer, riid, List of rgpszNames, cNames, Icid)	ACTIVE
31	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetIDsOfNames && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of rgDispld:=List of Unified Service Out-Parameter GetIDsOfNames.cnf(+)(Interface Pointer, hresult, List of rgDispld)	ACTIVE
32	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetIDsOfNames && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetIDsOfNames.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
33	ACTIVE	Invoke.req(Interface Pointer, dispIdMember, riid, lcId, wFlags, pDispParams) => Interface Pointer:= Interface Pointer Service Name:=Invoke List of Unified Service In-Parameter:=(dispIdMember, riid, lcId, wFlags, pDispParams) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
34	ACTIVE	Invoke.rsp(+)(Interface Pointer, hresult, pVarResult) => Interface Pointer:= Interface Pointer Service Name:=Invoke hresult:=hresult List of Unified Service Out-Parameter:=pVarResult ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
35	ACTIVE	Invoke.rsp(-)(Interface Pointer, hresult, pExcepInfo, puArgErr) => Interface Pointer:= Interface Pointer Service Name:=Invoke hresult:=hresult List of Unified Service Out-Parameter:=(pExcepInfo, puArgErr) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
36	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Invoke => Interface Pointer:=Interface Pointer dispIdMember:=List of Unified Service In-Parameter(dispIdMember) riid:=List of Unified Service In-Parameter(riid) lcId:=List of Unified Service In-Parameter(lcId) wFlags:=List of Unified Service In-Parameter(wFlags) pDispParams:=List of Unified Service In-Parameter(pDispParams) Invoke.ind(Interface Pointer, dispIdMember, riid, lcId, wFlags, pDispParams)	ACTIVE
37	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Invoke && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVarResult:=List of Unified Service Out-Parameter(pVarResult) Invoke.cnf(+)(Interface Pointer, hresult, pVarResult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
38	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Invoke && hresult.S=Error && hresult<>DISP_E_EXCEPTION && hresult<>DISP_E_TYPEMISMATCH && hresult<>DISP_E_PARAMNOTFOUND => Interface Pointer:=Interface Pointer hresult:=hresult pExcepInfo:="empty" puArgErr:="empty" Invoke.cnf(-)(Interface Pointer, hresult, pExcepInfo, puArgErr)	ACTIVE
39	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Invoke && (hresult=DISP_E_TYPEMISMATCH hresult=DISP_E_PARAMNOTFOUND) => Interface Pointer:=Interface Pointer hresult:=hresult pExcepInfo:="empty" puArgErr:=List of Unified Service Out-Parameter(puArgErr) Invoke.cnf(-)(Interface Pointer, hresult, pExcepInfo, puArgErr)	ACTIVE
40	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Invoke && hresult=DISP_E_EXCEPTION => Interface Pointer:=Interface Pointer hresult:=hresult pExcepInfo:=List of Unified Service Out-Parameter(pExcepInfo) puArgErr:="empty" Invoke.cnf(-)(Interface Pointer, hresult, pExcepInfo, puArgErr)	ACTIVE
41	ACTIVE	get_Producer.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=get_Producer List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
42	ACTIVE	get_Producer.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:= Interface Pointer Service Name:=get_Producer hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
43	ACTIVE	get_Producer.rsp(-)(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer Service Name:=get_Producer hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
44	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_Producer => Interface Pointer:=Interface Pointer get_Producer.ind(Interface Pointer)	ACTIVE
45	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Producer && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Producer.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
46	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Producer && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Producer.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
47	ACTIVE	get_Product.req(Interface Pointer) => Interface Pointer:= Interface Pointer Service Name:=get_Product List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
48	ACTIVE	get_Product.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:= Interface Pointer Service Name:=get_Product hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
49	ACTIVE	get_Product.rsp(-)(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer Service Name:=get_Product hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
50	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_Product => Interface Pointer:=Interface Pointer get_Product.ind(Interface Pointer)	ACTIVE
51	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Product&& hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Product.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
52	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Product && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Product.cnf(-)(Interface Pointer, hresult)	ACTIVE
53	ACTIVE	get_SerialNo.req(Interface Pointer) => Interface Pointer:= Interface Pointer Service Name:=get_SerialNo List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
54	ACTIVE	get_SerialNo.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:= Interface Pointer Service Name:=get_SerialNo hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
55	ACTIVE	get_SerialNo.rsp(-)(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer Service Name:=get_SerialNo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
56	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_SerialNo => Interface Pointer:=Interface Pointer get_SerialNo.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
57	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_SerialNo&& hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_SerialNo.cnf(+)(Interface Pointer,hresult, pVal)	ACTIVE
58	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_SerialNo && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_SerialNo.cnf(-)(Interface Pointer, hresult)	ACTIVE
59	ACTIVE	get_ProductionDate.req(Interface Pointer) => Interface Pointer:= Interface Pointer Service Name:=get_Product List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
60	ACTIVE	get_ProductionDate.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:=Interface Pointer Service Name:=get_ProductionDate hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
61	ACTIVE	get_ProductionDate.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_ProductionDate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
62	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_ProductionDate => Interface Pointer:=Interface Pointer get_ProductionDate.ind(Interface Pointer)	ACTIVE
63	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_ProductionDate && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_ProductionDate.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
64	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_ProductionDate && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_ProductionDate.cnf(-)(Interface Pointer, hresult)	ACTIVE
65	ACTIVE	Revision.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=Revision List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
66	ACTIVE	Revision.rsp(+)(Interface Pointer, hresult, pMajor, pMinor) => Interface Pointer:=Interface Pointer Service Name:=Revision hresult:=hresult List of Unified Service Out-Parameter:=(pMajor, pMinor) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
67	ACTIVE	Revision.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Revision hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
68	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Revision => Interface Pointer:=Interface Pointer Revision.ind(Interface Pointer)	ACTIVE
69	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Revision && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pMajor:=List of Unified Service Out-Parameter(pMajor) pMinor:=List of Unified Service Out-Parameter(pMinor) Revision.cnf(+)(Interface Pointer, hresult, pMajor, pMinor)	ACTIVE
70	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Revision && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Revision.cnf(-)(Interface Pointer, hresult)	ACTIVE
71	ACTIVE	get_LogicalDevice.req(Interface Pointer, Name) => Interface Pointer:= Interface Pointer Service Name:=get_LogicalDevice List of Unified Service In-Parameter:=(Name) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
72	ACTIVE	get_LogicalDevice.rsp(+)(Interface Pointer, hresult, ppLDev) => Interface Pointer:=Interface Pointer Service Name:=get_LogicalDevice hresult:=hresult List of Unified Service Out-Parameter:=ppLDev ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
73	ACTIVE	get_LogicalDevice.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_LogicalDevice hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
74	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_LogicalDevice => Interface Pointer:=Interface Pointer Name:=List of Unified Service In-Parameter get_LogicalDevice.ind(Interface Pointer, Name)	ACTIVE
75	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_LogicalDevice && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppLDev:=List of Unified Service Out-Parameter get_LogicalDevice.cnf(+)(Interface Pointer, hresult, ppLDev)	ACTIVE
76	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_LogicalDevice && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_LogicalDevice.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
77	ACTIVE	Type.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=Type List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
78	ACTIVE	Type.rsp(+)(Interface Pointer, hresult, pMultiApp, pPROFInetDcomStack) => Interface Pointer:=Interface Pointer Service Name:=Type hresult:=hresult List of Unified Service Out-Parameter:=(pMultiApp, pPROFInetDcomStack) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
79	ACTIVE	Type.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Type hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
80	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Type => Interface Pointer:=Interface Pointer Type.ind(Interface Pointer)	ACTIVE
81	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Type && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pMultiApp:=List of Unified Service Out-Parameter(pMultiApp) pPROFInetDcomStack:=List of Unified Service Out-Parameter(pPROFInetDcomStack) Type.cnf(+)(Interface Pointer, hresult, pMultiApp, pPROFInetDcomStack)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
82	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Type && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Type.cnf(-)(Interface Pointer, hresult)	ACTIVE
83	ACTIVE	PROFInetRevision.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=PROFInetRevision List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
84	ACTIVE	PROFInetRevision.rsp(+)(Interface Pointer, hresult, pMajor, pMinor, pServicePack, pBuild) => Interface Pointer:=Interface Pointer Service Name:=PROFInetRevision hresult:=hresult List of Unified Service Out-Parameter:=(pMajor, pMinor, pServicePack, pBuild) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
85	ACTIVE	PROFInetRevision.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=PROFInetRevision hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
86	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=PROFInetRevision => Interface Pointer:=Interface Pointer PROFInetRevision.cnf(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
87	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=PROFInetRevision && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pMajor:=List of Unified Service Out-Parameter(pMajor) pMinor:=List of Unified Service Out-Parameter(pMinor) pServicePack:=List of Unified Service Out-Parameter(pServicePack) pBuild:=List of Unified Service Out-Parameter(pBuild) PROFInetRevision.cnf(+)(Interface Pointer, hresult, pMajor, pMinor, pServicePack, pBuild)	ACTIVE
88	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=PROFInetRevision && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult PROFInetRevision.cnf(-)(Interface Pointer, hresult)	ACTIVE
89	ACTIVE	get_PDevStamp.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=get_PDevStamp List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
90	ACTIVE	get_PDevStamp.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:=Interface Pointer Service Name:=get_PDevStamp hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
91	ACTIVE	get_PDevStamp.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_PDevStamp hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
92	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_PDevStamp => Interface Pointer:=Interface Pointer get_PDevStamp.ind(Interface Pointer)	ACTIVE
93	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_PDevStamp && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_PDevStamp.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
94	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_PDevStamp && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_PDevStamp.cnf(-)(Interface Pointer, hresult)	ACTIVE
95	ACTIVE	get_Count.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=get_Count List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
96	ACTIVE	get_Count.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:=Interface Pointer Service Name:=get_Count hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
97	ACTIVE	get_Count.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_Count hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
98	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_Count => Interface Pointer:=Interface Pointer get_Count.ind(Interface Pointer)	ACTIVE
99	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Count && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Count.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
100	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Count && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Count.cnf(-)(Interface Pointer, hresult)	ACTIVE
101	ACTIVE	BrowseItems.req(Interface Pointer, Offset, MaxReturn) => Interface Pointer:=Interface Pointer Service Name:=BrowseItems List of Unified Service In-Parameter:=(Offset, MaxReturn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
102	ACTIVE	BrowseItems.rsp(+)(Interface Pointer, hresult, pltem, pDataType, pAccessRight) => Interface Pointer:=Interface Pointer Service Name:=BrowseItem hresult:=hresult List of Unified Service Out-Parameter:=(pltem, pDataType, pAccessRight) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
103	ACTIVE	BrowseItems.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=BrowseItems hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
104	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=BrowseItems => Interface Pointer:=Interface Pointer Offset:=List of Unified Service In-Parameter(Offset) MaxReturn:=List of Unified Service In-Parameter(MaxReturn) BrowseItem.ind(Interface Pointer, Offset, MaxReturn)	ACTIVE
105	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=BrowseItems && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pltem:=List of Unified Service Out-Parameter(pltem) pDataType:=List of Unified Service Out-Parameter(pDataType) pAccessRights:=List of Unified Service Out-Parameter(pAccessRights) BrowseItem.cnf(+)(Interface Pointer, hresult, pltem, pDataType, pAccessRight)	ACTIVE
106	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=BrowseItems && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult BrowseItem.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
107	ACTIVE	get_Count2.req(Interface Pointer, Selector) => Interface Pointer:=Interface Pointer Service Name:=get_Count2 List of Unified Service In-Parameter:=Selector ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
108	ACTIVE	get_Count2.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:=Interface Pointer Service Name:=get_Count2 hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
109	ACTIVE	get_Count2.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_Count2 hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
110	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_Count2 => Interface Pointer:=Interface Pointer Selector:=List of Unified Service In-Parameter get_Count2.ind(Interface Pointer, Selector)	ACTIVE
111	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Count2 && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Count2.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
112	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Count2 && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Count2.cnf(-)(Interface Pointer, hresult)	ACTIVE
113	ACTIVE	BrowseItems2.req(Interface Pointer, Selector, Offset, MaxReturn) => Interface Pointer:=Interface Pointer Service Name:=BrowseItems2 List of Unified Service In-Parameter:=(Selector, Offset, MaxReturn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
114	ACTIVE	BrowseItems2.rsp(+)(Interface Pointer, hresult, pItem, pInfo1, pInfo2) => Interface Pointer:=Interface Pointer Service Name:=BrowseItem2 hresult:=hresult List of Unified Service Out-Parameter:=(pItem, pInfo1, pInfo2) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
115	ACTIVE	BrowseItems2.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=BrowseItems2 hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
116	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=BrowseItems2 => Interface Pointer:=Interface Pointer Selector:=List of Unified Service In-Parameter(Selector) Offset:=List of Unified Service In-Parameter(Offset) MaxReturn:=List of Unified Service In-Parameter(MaxReturn) BrowseItems2.ind(Interface Pointer, Selector, Offset, MaxReturn)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
117	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=BrowseItems2 && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pltem:=List of Unified Service Out-Parameter(pltem) plnfo1:=List of Unified Service Out-Parameter(plnfo1) plnfo2:=List of Unified Service Out-Parameter(plnfo2) BrowseItems2.cnf(+)(Interface Pointer, hresult, pltem, plnfo1, plnfo2)	ACTIVE
118	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=BrowseItems2 && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult BrowseItems2.cnf(-)(Interface Pointer, hresult)	ACTIVE
119	ACTIVE	Save.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=Save List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
120	ACTIVE	Save.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Save hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
121	ACTIVE	Save.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Save hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
122	ACTIVE	ARPM_Call.ind (Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Save => Interface Pointer:=Interface Pointer Save.ind(Interface Pointer)	ACTIVE
123	ACTIVE	ARPM_Call.cnf (Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Save && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Save.cnf(+)(Interface Pointer, hresult)	ACTIVE
124	ACTIVE	ARPM_Call.cnf (Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Save && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Save.cnf(-)(Interface Pointer, hresult)	ACTIVE
125	ACTIVE	Save2.req (Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=Save2 List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
126	ACTIVE	Save2.rsp(+) (Interface Pointer, hresult, pLDevName, pPartialResult) => Interface Pointer:=Interface Pointer Service Name:=Save2 hresult:=hresult List of Unified Service Out-Parameter:=(pLDevName, pPartialResult) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
127	ACTIVE	Save2.rsp(-) (Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Save2 hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
128	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Save2 => Interface Pointer:=Interface Pointer Save2.ind(Interface Pointer)	ACTIVE
129	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Save2 && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pLDevName:=List of Unified Service Out-Parameter(pLDevName) pPartialResult:=List of Unified Service Out-Parameter(pPartialResult) Save2.cnf(+)(Interface Pointer, hresult, pLDevName, pPartialResult)	ACTIVE
130	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Save2 && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Save2.cnf(-)(Interface Pointer, hresult)	ACTIVE
131	ACTIVE	get_Name.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=get_Name List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
132	ACTIVE	get_Name.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:=Interface Pointer Service Name:=get_Name hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
133	ACTIVE	get_Name.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_Name hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
134	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_Name => Interface Pointer:=Interface Pointer get_Name.ind(Interface Pointer)	ACTIVE
135	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Name && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Name.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
136	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Name && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Name.cnf(-)(Interface Pointer, hresult)	ACTIVE
137	ACTIVE	get_ACCO.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=get_ACCO List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
138	ACTIVE	get_ACCO.rsp(+)(Interface Pointer, hresult, ppACCO) => Interface Pointer:=Interface Pointer Service Name:=get_ACCO hresult:=hresult List of Unified Service Out-Parameter:=ppACCO ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
139	ACTIVE	get_ACCO.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_ACCO hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
140	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_ACCO => Interface Pointer:=Interface Pointer get_ACCO.ind(Interface Pointer)	ACTIVE
141	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_ACCO && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppACCO:=List of Unified Service Out-Parameter get_ACCO.cnf(+)(Interface Pointer, hresult, ppACCO)	ACTIVE
142	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_ACCO && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_ACCO.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
143	ACTIVE	get_RTAuto.req(Interface Pointer, Name) => Interface Pointer:=Interface Pointer Service Name:=get_RTAuto List of Unified Service In-Parameter:=Name ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
144	ACTIVE	get_RTAuto.rsp(+)(Interface Pointer, hresult, ppAuto) => Interface Pointer:=Interface Pointer Service Name:=get_RTAuto hresult:=hresult List of Unified Service Out-Parameter:=ppAuto ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
145	ACTIVE	get_RTAuto.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_RTAuto hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
146	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_RTAuto => Interface Pointer:=Interface Pointer Name:=List of Unified Service InParameter get_RTAuto.ind(Interface Pointer, Name)	ACTIVE
147	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_RTAuto && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppAuto:=List of Unified Service Out-Parameter get_RTAuto.cnf(+)(Interface Pointer, hresult, ppAuto)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
148	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_RTAuto && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_RTAuto.cnf(-)(Interface Pointer, hresult)	ACTIVE
149	ACTIVE	ComponentInfo.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=ComponentInfo List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
150	ACTIVE	ComponentInfo.rsp(+)(Interface Pointer, hresult, pComponentID, pVersion) => Interface Pointer:=Interface Pointer Service Name:=ComponentInfo hresult:=hresult List of Unified Service Out-Parameter:=(pComponentID, pVersion) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
151	ACTIVE	ComponentInfo.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=ComponentInfo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
152	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=ComponentInfo => Interface Pointer:=Interface Pointer ComponentInfo.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
153	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ComponentInfo && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pComponentID:=List of Unified Service Out-Parameter(pComponentID) pVersion:=List of Unified Service Out-Parameter(pVersion) ComponentInfo.cnf(+)(Interface Pointer, hresult, pComponentID, pVersion)	ACTIVE
154	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ComponentInfo && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult ComponentInfo.cnf(-)(Interface Pointer, hresult)	ACTIVE
155	ACTIVE	get_State.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=get_State List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
156	ACTIVE	get_State.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:=Interface Pointer Service Name:=get_State hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
157	ACTIVE	get_State.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_State hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
158	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_State => Interface Pointer:=Interface Pointer get_State.ind(Interface Pointer)	ACTIVE
159	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_State && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_State.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
160	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_State && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_State.cnf(-)(Interface Pointer, hresult)	ACTIVE
161	ACTIVE	Activate.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=Activate List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
162	ACTIVE	Activate.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Activate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
163	ACTIVE	Activate.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Activate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
164	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Activate => Interface Pointer:=Interface Pointer Activate.ind(Interface Pointer)	ACTIVE
165	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Activate && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Activate.cnf(+)(Interface Pointer, hresult)	ACTIVE
166	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Activate && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Activate.cnf(-)(Interface Pointer, hresult)	ACTIVE
167	ACTIVE	Deactivate.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=Deactivate List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
168	ACTIVE	Deactivate.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Deactivate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
169	ACTIVE	Deactivate.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Deactivate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
170	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Deactivate => Interface Pointer:=Interface Pointer Deactivate.ind(Interface Pointer)	ACTIVE
171	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Deactivate && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Deactivate.cnf(+)(Interface Pointer, hresult)	ACTIVE
172	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Deactivate && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Deactivate.cnf(-)(Interface Pointer, hresult)	ACTIVE
173	ACTIVE	Reset.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=Reset List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
174	ACTIVE	Reset.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Reset hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
175	ACTIVE	Reset.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Reset hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
176	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Reset => Interface Pointer:=Interface Pointer Reset.ind(Interface Pointer)	ACTIVE
177	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Reset && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Reset.cnf(+)(Interface Pointer, hresult)	ACTIVE
178	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Reset && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Reset.cnf(-)(Interface Pointer, hresult)	ACTIVE
179	ACTIVE	AdviseState.req(Interface Pointer, pStateEvent) => Interface Pointer:=Interface Pointer Service Name:=AdviseState List of Unified Service In-Parameter:=pStateEvent ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
180	ACTIVE	AdviseState.rsp(+)(Interface Pointer, hresult, pCookie) => Interface Pointer:=Interface Pointer Service Name:=AdviseState hresult:=hresult List of Unified Service Out-Parameter:=pCookie ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
181	ACTIVE	AdviseState.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=AdviseState hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
182	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=AdviseState => Interface Pointer:=Interface Pointer pStateEvent:=List of Unified Service In-Parameter AdviseState.ind(Interface Pointer, pStateEvent)	ACTIVE
183	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=AdviseState && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pCookie:=List of Unified Service Out-Parameter AdviseState.cnf(+)(Interface Pointer, hresult, pCookie)	ACTIVE
184	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=AdviseState && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult AdviseState.cnf(-)(Interface Pointer, hresult)	ACTIVE
185	ACTIVE	UnadviseState.req(Interface Pointer, Cookie) => Interface Pointer:=Interface Pointer Service Name:=UnadviseState List of Unified Service In-Parameter:=Cookie ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
186	ACTIVE	UnadviseState.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=UnadviseState hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
187	ACTIVE	UnadviseState.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=UnadviseState hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
188	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=UnadviseState => Interface Pointer:=Interface Pointer Cookie:=List of Unified Service In-Parameter UnadviseState.ind(Interface Pointer, Cookie)	ACTIVE
189	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=UnadviseState && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult UnadviseState.cnf(+)(Interface Pointer, hresult)	ACTIVE
190	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=UnadviseState && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult UnadviseState.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
191	ACTIVE	get_Time.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=get_Time List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
192	ACTIVE	get_Time.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:=Interface Pointer Service Name:=get_Time hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
193	ACTIVE	get_Time.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_Time hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
194	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_Time => Interface Pointer:=Interface Pointer get_Time.ind(Interface Pointer)	ACTIVE
195	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Time && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Time.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
196	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_Time && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Time.cnf(-)(Interface Pointer, hresult)	ACTIVE
197	ACTIVE	put_Time.req(Interface Pointer, NewVal) => Interface Pointer:=Interface Pointer Service Name:=put_Time List of Unified Service In-Parameter:=NewVal ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
198	ACTIVE	put_Time.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=put_Time hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
199	ACTIVE	put_Time.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=put_Time hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
200	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=put_Time => Interface Pointer:=Interface Pointer NewVal:=List of Unified Service In-Parameter put_Time.ind(Interface Pointer, NewVal)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
201	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=put_Time && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult put_Time.cnf(+)(Interface Pointer, hresult)	ACTIVE
202	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=put_Time && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult put_Time.cnf(-)(Interface Pointer, hresult)	ACTIVE
203	ACTIVE	GroupError.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=GroupError List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
204	ACTIVE	GroupError.rsp(+)(Interface Pointer, hresult, pVal, pMagicCookie) => Interface Pointer:=Interface Pointer Service Name:=GroupError hresult:=hresult List of Unified Service Out-Parameter:=(pVal, pMagicCookie) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
205	ACTIVE	GroupError.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GroupError hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
206	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GroupError => Interface Pointer:=Interface Pointer GroupError.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
207	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GroupError && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter(pVal) pMagicCookie:=List of Unified Service Out-Parameter(pMagicCookie) GroupError.cnf(+)(Interface Pointer, hresult, pVal, pMagicCookie)	ACTIVE
208	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GroupError && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GroupError.cnf(-)(Interface Pointer, hresult)	ACTIVE
209	ACTIVE	AdviseGroupError.req(Interface Pointer, pGroupErrorEvent) => Interface Pointer:=Interface Pointer Service Name:=AdviseGroupError List of Unified Service In-Parameter:=pGroupErrorEvent ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
210	ACTIVE	AdviseGroupError.rsp(+)(Interface Pointer, hresult, pCookie) => Interface Pointer:=Interface Pointer Service Name:=AdviseGroupError hresult:=hresult List of Unified Service Out-Parameter:=pCookie ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
211	ACTIVE	AdviseGroupError.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=AdviseGroupError hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
212	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=AdviseGroupError => Interface Pointer:=Interface Pointer pGroupErrorEvent:=List of Unified Service In-Parameter AdviseGroupError.ind(Interface Pointer, pGroupErrorEvent)	ACTIVE
213	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=AdviseGroupError && hresult.S=Success S_FALSE => Interface Pointer:=Interface Pointer hresult:=hresult pCookie:=List of Unified Service Out-Parameter AdviseGroupError.cnf(+)(Interface Pointer, hresult, pCookie)	ACTIVE
214	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=AdviseGroupError && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult AdviseGroupError.cnf(-)(Interface Pointer, hresult)	ACTIVE
215	ACTIVE	UnadviseGroupError.req(Interface Pointer, Cookie) => Interface Pointer:=Interface Pointer Service Name:=UnadviseGroupError List of Unified Service In-Parameter:=Cookie ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
216	ACTIVE	UnadviseGroupError.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=UnadviseGroupError hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
217	ACTIVE	UnadviseGroupError.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=UnadviseGroupError hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
218	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=UnadviseGroupError => Interface Pointer:=Interface Pointer Cookie:=List of Unified Service In-Parameter UnadviseGroupError.ind(Interface Pointer, Cookie)	ACTIVE
219	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=UnadviseGroupError && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult UnadviseGroupError.cnf(+)(Interface Pointer, hresult)	ACTIVE
220	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=UnadviseGroupError && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult UnadviseGroupError.cnf(-)(Interface Pointer, hresult)	ACTIVE
221	ACTIVE	AddConnections.req(Interface Pointer, Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn) => Interface Pointer:=Interface Pointer Service Name:=AddConnections List of Unified Service In-Parameter:=(Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
222	ACTIVE	AddConnections.rsp(+)(Interface Pointer, hresult, List of ppAddConnectionOut) => Interface Pointer:=Interface Pointer Service Name:=AddConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppAddConnectionOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
223	ACTIVE	AddConnections.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=AddConnection hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
224	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=AddConnections => Interface Pointer:=Interface Pointer Provider:=List of Unified Service In-Parameter(Provider) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) State:=List of Unified Service In-Parameter(State) Count:=List of Unified Service In-Parameter(Count) List of pAddConnectionIn:=List of Unified Service In-Parameter(List of pAddConnectionIn) AddConnections.ind(Interface Pointer, Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn)	ACTIVE
225	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=AddConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppAddConnectionOut:=List of Unified Service Out-Parameter AddConnections.cnf(+)(Interface Pointer, hresult, List of ppAddConnectionOut)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
226	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=AddConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult AddConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
227	ACTIVE	RemoveConnections.req(Interface Pointer, Count, List of pConsumerID) => Interface Pointer:=Interface Pointer Service Name:=RemoveConnections List of Unified Service In-Parameter:=(Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
228	ACTIVE	RemoveConnections.rsp(+)(Interface Pointer, hresult, List of ppError) => Interface Pointer:=Interface Pointer Service Name:=RemoveConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
229	ACTIVE	RemoveConnections.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=RemoveConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
230	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=RemoveConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameterer(List of pConsumerID) RemoveConnections.ind(Interface Pointer, Count, List of pConsumerID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
231	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=RemoveConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter RemoveConnections.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
232	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=RemoveConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult RemoveConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
233	ACTIVE	ClearConnections.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=ClearConnections List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
234	ACTIVE	ClearConnections.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=ClearConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
235	ACTIVE	ClearConnections.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=ClearConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
236	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=ClearConnections => Interface Pointer:=Interface Pointer ClearConnections.ind(Interface Pointer)	ACTIVE
237	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ClearConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ClearConnections.cnf(+)(Interface Pointer, hresult)	ACTIVE
238	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ClearConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult ClearConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
239	ACTIVE	SetActivationState.req(Interface Pointer, State, Count, List of pConsumerID) => Interface Pointer:=Interface Pointer Service Name:=SetActivationState List of Unified Service In-Parameter:=(State, Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
240	ACTIVE	SetActivationState.rsp(+)(Interface Pointer, hresult, List of ppError) => Interface Pointer:=Interface Pointer Service Name:=SetActivationState hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
241	ACTIVE	SetActivationState.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=SetActivationState hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
242	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=SetActivationState => Interface Pointer:=Interface Pointer State:=List of Unified Service In-Parameter(State) Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameter(List of pConsumerID) SetActivationState.ind(Interface Pointer, State, Count, List of pConsumerID)	ACTIVE
243	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=SetActivationState && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter SetActivationState.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
244	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=AddConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult SetActivationState.cnf(-)(Interface Pointer, hresult)	ACTIVE
245	ACTIVE	GetInfo.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=GetInfo List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
246	ACTIVE	GetInfo.rsp(+)(Interface Pointer, hresult, pMax, pCurCnt) => Interface Pointer:=Interface Pointer Service Name:=GetInfo hresult:=hresult List of Unified Service Out-Parameter:=(pMax, pCurCnt) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
247	ACTIVE	GetInfo.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GetInfo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
248	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetInfo => Interface Pointer:=Interface Pointer GetInfo.ind(Interface Pointer)	ACTIVE
249	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetInfo && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pMax:=List of Unified Service Out-Parameter(pMax) pCurCnt:=List of Unified Service Out-Parameter(pCurCnt) GetInfo.cnf(+)(Interface Pointer, hresult, pMax, pCurCnt)	ACTIVE
250	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetInfo && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetInfo.cnf(-)(Interface Pointer, hresult)	ACTIVE
251	ACTIVE	GetIDs.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=GetIDs List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
252	ACTIVE	GetIDs.rsp(+)(Interface Pointer, hresult, pCount, List of ppGetIDOut) => Interface Pointer:=Interface Pointer Service Name:=GetIDs hresult:=hresult List of Unified Service Out-Parameter:=(pCount, List of ppGetIDOut) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
253	ACTIVE	GetIDs.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GetInfo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
254	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetIDs => Interface Pointer:=Interface Pointer GetIDs.ind(Interface Pointer)	ACTIVE
255	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetIDs && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pCount:=List of Unified Service Out-Parameter(pCount) List of ppGetIDOut:=List of Unified Service Out-Parameter(List of ppGetIDOut) GetIDs.cnf(+)(Interface Pointer, hresult, pCount, List of ppGetIDOut)	ACTIVE
256	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetIDs && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetIDs.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
257	ACTIVE	GetConnections.req(Interface Pointer, Count, List of pConsumerID) => Interface Pointer:=Interface Pointer Service Name:=GetConnections List of Unified Service In-Parameter:=(Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
258	ACTIVE	GetConnections.rsp(+)(Interface Pointer, hresult, List of ppGetConnectionOut) => Interface Pointer:=Interface Pointer Service Name:=GetConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppGetConnectionOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
259	ACTIVE	GetConnections.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GetConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
260	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameter(List of pConsumerID) GetConnections.ind(Interface Pointer, Count, List of pConsumerID)	ACTIVE
261	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppGetConnectionsOut:=List of Unified Service Out-Parameter GetConnections.cnf(+)(Interface Pointer, hresult, List of ppGetConnectionsOut)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
262	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
263	ACTIVE	ReviseQoS.req(Interface Pointer, RTAuto, QoSType, QoSValue) => Interface Pointer:=Interface Pointer Service Name:=ReviseQoS List of Unified Service In-Parameter:=(RTAuto, QoSType, QoSValue) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
264	ACTIVE	ReviseQoS.rsp(+)(Interface Pointer, hresult, pRevisedQoSValue) => Interface Pointer:=Interface Pointer Service Name:=ReviseQoS hresult:=hresult List of Unified Service Out-Parameter:=ppRevisedQoSValue ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
265	ACTIVE	ReviseQoS.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=ReviseQoS hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
266	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=ReviseQoS => Interface Pointer:=Interface Pointer RTAuto:=List of Unified Service In-Parameter(RTAuto) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) ReviseQoS.ind(Interface Pointer, RTAuto, QoSType, QoSValue)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
267	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=RevisedQoS && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pRevisedQoSValue:=List of Unified Service Out-Parameter RevisedQoS.cnf(+)(Interface Pointer, hresult, pRevisedQoSValue)	ACTIVE
268	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=RevisedQoS && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult RevisedQoS.cnf(-)(Interface Pointer, hresult)	ACTIVE
269	ACTIVE	get_PingFactor.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=get_PingFactor List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
270	ACTIVE	get_PingFactor.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:=Interface Pointer Service Name:=get_PingFactor hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
271	ACTIVE	get_PingFactor.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_PingFactor hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
272	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_PingFactor => Interface Pointer:=Interface Pointer get_PingFactor.ind(Interface Pointer)	ACTIVE
273	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_PingFactor && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_PingFactor.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
274	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_PingFactor && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_PingFactor.cnf(-)(Interface Pointer, hresult)	ACTIVE
275	ACTIVE	put_PingFactor.req(Interface Pointer, NewVal) => Interface Pointer:=Interface Pointer Service Name:=put_PingFactor List of Unified Service In-Parameter:=NewVal ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
276	ACTIVE	put_PingFactor.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=put_PingFactor hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
277	ACTIVE	put_PingFactor.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=put_PingFactor hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
278	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=put_PingFactor => Interface Pointer:=Interface Pointer NewVal:=List of Unified Service In-Parameter put_PingFactor.ind(Interface Pointer, NewVal)	ACTIVE
279	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=put_PingFactor && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult put_PingFactor.cnf(+)(Interface Pointer, hresult)	ACTIVE
280	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=put_PingFactor && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult put_PingFactor.cnf(-)(Interface Pointer, hresult)	ACTIVE
281	ACTIVE	get_CDBCcookie.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=get_CDBCcookie List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
282	ACTIVE	get_CDBCcookie.rsp(+)(Interface Pointer, hresult, pVal) => Interface Pointer:=Interface Pointer Service Name:=get_CDBCcookie hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
283	ACTIVE	get_CDBCcookie.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=get_CDBCcookie hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
284	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=get_CDBCcookie => Interface Pointer:=Interface Pointer get_CDBCcookie.ind(Interface Pointer)	ACTIVE
285	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_CDBCcookie && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_CDBCcookie.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
286	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=get_CDBCcookie && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_CDBCcookie.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
287	ACTIVE	GetConsIDs.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=GetConsIDs List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
288	ACTIVE	GetConsIDs.rsp(+)(Interface Pointer, hresult, pCount, List of ppConsumerID) => Interface Pointer:=Interface Pointer Service Name:=GetConsIDs hresult:=hresult List of Unified Service Out-Parameter:=(pCount, List of ppConsumerID) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
289	ACTIVE	GetConsIDs.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GetConsIDs hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
290	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetConsIDs => Interface Pointer:=Interface Pointer GetConsIDs.ind(Interface Pointer)	ACTIVE
291	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetConsIDs && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pCount:=List of Unified Service Out-Parameter(pCount) List of ppConsumerID:=List of Unified Service Out-Parameter(List of ppConsumerID) GetConsIDs.cnf(+)(Interface Pointer, hresult, pCount, List of ppConsumerID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
292	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetConsIDs && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetConsIDs.cnf(-)(Interface Pointer, hresult)	ACTIVE
293	ACTIVE	GetConsConnections.req(Interface Pointer, Count, List of pConsumerID) => Interface Pointer:=Interface Pointer Service Name:=GetConsConnections List of Unified Service In-Parameter:=(Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
294	ACTIVE	GetConsConnections.rsp(+)(Interface Pointer, hresult, List of ppGetConsConnOut) => Interface Pointer:=Interface Pointer Service Name:=GetConsConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppGetConsConnOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
295	ACTIVE	GetConsConnections.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GetConsConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
296	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetConsConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameter(List of pConsumerID) GetConsConnections.ind(Interface Pointer, Count, List of pConsumerID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
297	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetConsConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppGetConsConnOut:=List of Unified Service Out-Parameter GetConsConnections.cnf(+)(Interface Pointer, hresult, List of ppGetConsConnOut)	ACTIVE
298	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetConsConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetConsConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
299	ACTIVE	DiagConsConnections.req(Interface Pointer, Count, List of pConsumerID) => Interface Pointer:=Interface Pointer Service Name:=DiagConsConnections List of Unified Service In-Parameter:=(Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
300	ACTIVE	DiagConsConnections.rsp(+)(Interface Pointer, hresult, List of ppDiagConsConnOut) => Interface Pointer:=Interface Pointer Service Name:=DiagConsConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppDiagConsConnOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
301	ACTIVE	DiagConsConnections.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=DiagConsConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
302	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=DiagConsConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameter(List of pConsumerID) DiagConsConnections.ind(Interface Pointer, Count, List of pConsumerID)	ACTIVE
303	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=DiagConsConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppDiagConsConnOut:=List of Unified Service Out-Parameter DiagConsConnections.cnf(+) (Interface Pointer, hresult, List of ppDiagConsConnOut)	ACTIVE
304	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=DiagConsConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult DiagConsConnections.cnf(-) (Interface Pointer, hresult)	ACTIVE
305	ACTIVE	GetProvIDs.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=GetProvIDs List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
306	ACTIVE	GetProvIDs.rsp(+)(Interface Pointer, hresult, pCount, List of ppProviderID) => Interface Pointer:=Interface Pointer Service Name:=GetProvIDs hresult:=hresult List of Unified Service Out-Parameter:=(pCount, List of ppProviderID) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
307	ACTIVE	GetProvIDs.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GetProvIDs hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
308	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetProvIDs => Interface Pointer:=Interface Pointer GetProvIDs.ind(Interface Pointer)	ACTIVE
309	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetProvIDs && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pCount:=List of Unified Service Out-Parameter(pCount) List of ppProviderID:=List of Unified Service Out-Parameter(List of ppProviderID) GetProvIDs.cnf(+)(Interface Pointer, hresult, pCount, List of ppProviderID)	ACTIVE
310	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetProvIDs && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetProvIDs.cnf(-)(Interface Pointer, hresult)	ACTIVE
311	ACTIVE	GetProvConnections.req(Interface Pointer, Count, List of pProviderID) => Interface Pointer:=Interface Pointer Service Name:=GetProvConnections List of Unified Service In-Parameter:=(Count, List of pProviderID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
312	ACTIVE	GetProvConnections.rsp(+)(Interface Pointer, hresult, List of ppGetProvConnOut) => Interface Pointer:=Interface Pointer Service Name:=GetProvConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppGetProvConnOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
313	ACTIVE	GetProvConnections.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GetProvConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
314	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetProvConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pProviderID:=List of Unified Service In-Parameter(List of pProviderID) GetProvConnections.ind(Interface Pointer, Count, List of pProviderID)	ACTIVE
315	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetProvConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppDiagConsConnOut:=List of Unified Service Out-Parameter GetProvConnections.cnf(+)(Interface Pointer, hresult, List of ppGetProvConnOut)	ACTIVE
316	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetProvConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetProvConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
317	ACTIVE	GetDiagnosis.req(Interface Pointer, Request, InLength, pInBuffer) => Interface Pointer:=Interface Pointer Service Name:=GetDiagnosis List of Unified Service In-Parameter:=(Request, InLength, pInBuffer) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
318	ACTIVE	GetDiagnosis.rsp(+)(Interface Pointer, hresult, pOutLength, ppOutBuffer) => Interface Pointer:=Interface Pointer Service Name:=GetDiagnosis hresult:=hresult List of Unified Service Out-Parameter:=(pOutLength, ppOutBuffer) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
319	ACTIVE	GetDiagnosis.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GetDiagnosis hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
320	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetDiagnosis => Interface Pointer:=Interface Pointer Request:=List of Unified Service In-Parameter(Request) InLength:=List of Unified Service In-Parameter(InLength) pInBuffer:=List of Unified Service In-Parameter(pInBuffer) GetDiagnosis.ind(Interface Pointer, Request, InLength, pInBuffer)	ACTIVE
321	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetDiagnosis && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pOutLength:=List of Unified Service Out-Parameter(pOutLength) ppOutBuffer:=List of Unified Service Out-Parameter(ppOutBuffer) GetDiagnosis.cnf(+)(Interface Pointer, hresult, pOutLength, ppOutBuffer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
322	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetDiagnosis && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetDiagnosis.cnf(-)(Interface Pointer, hresult)	ACTIVE
323	ACTIVE	Connect.req(Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn) => Interface Pointer:=Interface Pointer Service Name:=Connect List of Unified Service In-Parameter:=(Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
324	ACTIVE	Connect.rsp(+)(Interface Pointer, hresult, pFirstConnect, List of ppConnectOut) => Interface Pointer:=Interface Pointer Service Name:=Connect hresult:=hresult List of Unified Service Out-Parameter:=(pFirstConnect, List of ppConnectOut) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
325	ACTIVE	Connect.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Connect hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
326	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Connect => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter(Consumer) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) State:=List of Unified Service In-Parameter(State) pICBACallback:=List of Unified Service In-Parameter(pICBACallback) Count:=List of Unified Service In-Parameter(Count) List of pConnectIn:=List of Unified Service In-Parameter(List of pConnectIn) Connect.ind(Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAccoCallBack, Count, List of pConnectIn)	ACTIVE
327	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Connect && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pFirstConnect:=List of Unified Service Out-Parameter(pFirstConnect) List of ppConnectOut:=List of Unified Service Out-Parameter(List of ppConnectOut) Connect.cnf(+)(Interface Pointer, hresult, pFirstConnect, List of ppConnectOut)	ACTIVE
328	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Connect && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Connect.cnf(-)(Interface Pointer, hresult)	ACTIVE
329	ACTIVE	Disconnect.req(Interface Pointer, Count, List of pProviderID) => Interface Pointer:=Interface Pointer Service Name:=Disconnect List of Unified Service In-Parameter:=(Count, List of pProviderID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
330	ACTIVE	Disconnect.rsp(+)(Interface Pointer, hresult, List of ppError) => Interface Pointer:=Interface Pointer Service Name:=Disconnect hresult:=hresult List of Unified Service Out-Parameter:=(List of ppError) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
331	ACTIVE	Disconnect.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Disconnect hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
332	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Disconnect => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pProviderID:=List of Unified Service In-Parameter(List of pProviderID) Disconnect.ind(Interface Pointer, Count, List of pProviderID)	ACTIVE
333	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Disconnect && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter(List of ppError) Disconnect.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
334	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Disconnect && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Disconnect.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
335	ACTIVE	DisconnectMe.req(Interface Pointer, Consumer) => Interface Pointer:=Interface Pointer Service Name:=DisconnectMe List of Unified Service In-Parameter:=Consumer ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
336	ACTIVE	DisconnectMe.rsp(+)(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer Service Name:=DisconnectMe hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
337	ACTIVE	DisconnectMe.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=DisconnectMe hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
338	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=DisconnectMe => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter DisconnectMe.ind(Interface Pointer, Consumer)	ACTIVE
339	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=DisconnectMe && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult DisconnectMe.cnf(+)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
340	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=DisconnectMe && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult DisconnectMe.cnf(-)(Interface Pointer, hresult)	ACTIVE
341	ACTIVE	SetActivation.req(Interface Pointer, State, Count, List of pProviderID) => Interface Pointer:= Interface Pointer Service Name:=SetActivation List of Unified Service In-Parameter:=(State, Count, List of pProviderID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
342	ACTIVE	SetActivation.rsp(+)(Interface Pointer, hresult, List of ppError) => Interface Pointer:=Interface Pointer Service Name:=SetActivation hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
343	ACTIVE	SetActivation.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=SetActivation hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
344	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=SetActivation => Interface Pointer:=Interface Pointer State:=List of Unified Service In-Parameter(State) Count:=List of Unified Service In-Parameter(Count) List of pProviderID:=List of Unified Service In-Parameter(List of pProviderID) SetActivation.ind(Interface Pointer, State, Count, List of pProviderID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
345	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=SetActivation && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter SetActivation.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
346	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=SetActivation && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult SetActivation.cnf(-)(Interface Pointer, hresult)	ACTIVE
347	ACTIVE	Ping.req(Interface Pointer, Consumer) => Interface Pointer:=Interface Pointer Service Name:=Ping List of Unified Service In-Parameter:=Consumer ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
348	ACTIVE	Ping.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Ping hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
349	ACTIVE	Ping.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Ping hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
350	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Ping => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter Ping.ind(Interface Pointer, Consumer)	ACTIVE
351	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Ping && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Ping.cnf(+)(Interface Pointer, hresult)	ACTIVE
352	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Ping && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Ping.cnf(-)(Interface Pointer, hresult)	ACTIVE
353	ACTIVE	Connect2.req(Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn) => Interface Pointer:=Interface Pointer Service Name:=Connect2 List of Unified Service In-Parameter:=(Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
354	ACTIVE	Connect2.rsp(+)(Interface Pointer, hresult, pFirstConnect, List of ppConnectOut) => Interface Pointer:=Interface Pointer Service Name:=Connect2 hresult:=hresult List of Unified Service Out-Parameter:=(pFirstConnect, List of ppConnectOut) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
355	ACTIVE	Connect2.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Connect2 hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
356	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Connect2 => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter(Consumer) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) State:=List of Unified Service In-Parameter(State) pICBACallback:=List of Unified Service In-Parameter(pICBACallback) Count:=List of Unified Service In-Parameter(Count) List of pConnectIn:=List of Unified Service In-Parameter(List of pConnectIn) Connect2.ind(Interface Pointer, Consumer, QoSType, QoSValue, State, pICBACallback, Count, List of pConnectIn)	ACTIVE
357	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Connect2 && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pFirstConnect:=List of Unified Service Out-Parameter(pFirstConnect) List of ppConnectOut:=List of Unified Service Out-Parameter(List of ppConnectOut) Connect2.cnf(+)(Interface Pointer, hresult, pFirstConnect, List of ppConnectOut)	ACTIVE
358	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Connect2 && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Connect2.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
359	ACTIVE	GetConnectionData.req(Interface Pointer, Consumer) => Interface Pointer:=Interface Pointer Service Name:=GetConnectionData List of Unified Service In-Parameter:=Consumer ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
360	ACTIVE	GetConnectionData.rsp(+)(Interface Pointer, hresult, pLength, pBuffer) => Interface Pointer:=Interface Pointer Service Name:=GetConnectionData hresult:=hresult List of Unified Service Out-Parameter:=(pLength, pBuffer) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
361	ACTIVE	GetConnectionData.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=GetConnectionData hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
362	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=GetConnectionData => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter GetConnectionData.ind(Interface Pointer, Consumer)	ACTIVE
363	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetConnectionData && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pLength:=List of Unified Service In-Parameter(pLength) pBuffer:=List of Unified Service In-Parameter(pBuffer) GetConnectionData.cnf(+)(Interface Pointer, hresult, pLength, pBuffer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
364	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=GetConnectionData && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetConnectionData.cnf(-)(Interface Pointer, hresult)	ACTIVE
365	ACTIVE	OnDataChanged.req(Interface Pointer, Length, pBuffer) => Interface Pointer:=Interface Pointer Service Name:=OnDataChanged List of Unified Service In-Parameter:=(Length, pBuffer) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
366	ACTIVE	OnDataChanged.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=OnDataChanged hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
367	ACTIVE	OnDataChanged.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=OnDataChanged hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
368	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=OnDataChanged => Interface Pointer:=Interface Pointer Length:=List of Unified Service In-Parameter(Length) pBuffer:=List of Unified Service In-Parameter(pBuffer) OnDataChanged.ind(Interface Pointer, Length, pBuffer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
369	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=OnDataChanged && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult OnDataChanged.cnf(+)(Interface Pointer, hresult)	ACTIVE
370	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=OnDataChanged && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult OnDataChanged.cnf(-)(Interface Pointer, hresult)	ACTIVE
371	ACTIVE	Gnip.req(Interface Pointer) => Interface Pointer:=Interface Pointer Service Name:=Gnip List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
372	ACTIVE	Gnip.rsp(+)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Gnip hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
373	ACTIVE	Gnip.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Gnip hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
374	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Gnip => Interface Pointer:=Interface Pointer Gnip.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
375	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Gnip && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Gnip.cnf(+)(Interface Pointer, hresult)	ACTIVE
376	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Gnip && hresult.S=Errors => Interface Pointer:=Interface Pointer hresult:=hresult Gnip.cnf(-)(Interface Pointer, hresult)	ACTIVE
377	ACTIVE	ReadItems.req(Interface Pointer, Count, List of pReadItem) => Interface Pointer:=Interface Pointer Service Name:=ReadItems List of Unified Service In-Parameter:=(Count, List of pReadItem) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
378	ACTIVE	ReadItems.rsp(+)(Interface Pointer, hresult, List of ppReadItemOut) => Interface Pointer:=Interface Pointer Service Name:=ReadItems hresult:=hresult List of Unified Service Out-Parameter:=List of ppReadItemOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
379	ACTIVE	ReadItems.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=ReadItems hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
380	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=ReadItems => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pReadItem:=List of Unified Service In-Parameter(List of pReadItem) ReadItems.ind(Interface Pointer, Count, List of pReadItem)	ACTIVE
381	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ReadItems && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppReadItemOut:=List of Unified Service Out-Parameter ReadItems.cnf(+)(Interface Pointer, hresult, List of ppReadItemOut)	ACTIVE
382	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ReadItems && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult ReadItems.cnf(-)(Interface Pointer, hresult)	ACTIVE
383	ACTIVE	WriteItems.req(Interface Pointer, Count, List of pWriteItem) => Interface Pointer:=Interface Pointer Service Name:=WriteItems List of Unified Service In-Parameter:=(Count, List of pWriteItem) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
384	ACTIVE	WriteItems.rsp(+)(Interface Pointer, hresult, List of ppError) => Interface Pointer:=Interface Pointer Service Name:=WriteItems hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
385	ACTIVE	Writeltems.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=Writeltems hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
386	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=Writeltems => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pWriteltem:=List of Unified Service In-Parameter(List of pWriteltem) Writeltems.ind(Interface Pointer, Count, List of pWriteltem)	ACTIVE
387	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Writeltems && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter Writeltems.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
388	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=Writeltems && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Writeltems.cnf(-)(Interface Pointer, hresult)	ACTIVE
389	ACTIVE	WriteltemsQCD.req(Interface Pointer, Count, List of pWriteltemQCDIn) => Interface Pointer:=Interface Pointer Service Name:=WriteltemsQCD List of Unified Service In-Parameter:=(Count, List of pWriteltemQCDIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
390	ACTIVE	WriteItemsQCD.rsp(+)(Interface Pointer, hresult, List of ppError) => Interface Pointer:=Interface Pointer Service Name:=WriteItemsQCD hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
391	ACTIVE	WriteItemsQCD.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=ReadItemsQCD hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
392	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=WriteItemsQCD => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pWriteItemQCdIn:=List of Unified Service In-Parameter(List of pWriteItemQCdIn) WriteItemsQCD.ind(Interface Pointer, Count, List of pWriteItemQCdIn)	ACTIVE
393	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=WriteItemsQCD && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter WriteItemsQCD.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
394	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=WriteItemsQCD && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult WriteItemsQCD.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
395	ACTIVE	ConnectCR.req(Interface Pointer, Consumer, QoSType, QoSValue, pICBAAccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn) => Interface Pointer:=Interface Pointer Service Name:=ConnectCR List of Unified Service In-Parameter:=(Consumer, QoSType, QoSValue, pICBAAccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
396	ACTIVE	ConnectCR.rsp(+)(Interface Pointer, hresult, pFirstConnect, pProviderMAC, List of ppConnectOut) => Interface Pointer:=Interface Pointer Service Name:=ConnectCR hresult:=hresult List of Unified Service Out-Parameter:=(pFirstConnect, pProviderMAC, List of ppConnectOut) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
397	ACTIVE	ConnectCR.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=ConnectCR hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
398	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=ConnectCR => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter(Consumer) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) pICBACallback:=List of Unified Service In-Parameter(pICBACallback) ConsumerMAC:=List of Unified Service In-Parameter(ConsumerMAC) Flags:=List of Unified Service In-Parameter(Flags) Count:=List of Unified Service In-Parameter(Count) List of pConnectIn:=List of Unified Service In-Parameter(List of pConnectIn) ConnectCR.ind(Interface Pointer, Consumer, QoSType, QoSValue, pICBAAccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
399	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ConnectCR && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pFirstConnect:=List of Unified Service Out-Parameter(pFirstConnect) pProviderMAC:=List of Unified Service Out-Parameter(pProviderMAC) List of ppConnectOut:=List of Unified Service Out-Parameter(List of ppConnectOut) ConnectCR.cnf(+)(Interface Pointer, hresult, pFirstConnect, pProviderMAC, List of ppConnectOut)	ACTIVE
400	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ConnectCR && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult ConnectCR.cnf(-)(Interface Pointer, hresult)	ACTIVE
401	ACTIVE	DisconnectCR.req(Interface Pointer, Count, List of pProviderCRID) => Interface Pointer:=Interface Pointer Service Name:=DisconnectCR List of Unified Service In-Parameter:=(Count, List of pProviderCRID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
402	ACTIVE	DisconnectCR.rsp(+)(Interface Pointer, hresult, List of ppError) => Interface Pointer:=Interface Pointer Service Name:=DisconnectCR hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
403	ACTIVE	DisconnectCR.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=DisconnectCR hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
404	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=DisconnectCR => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pProviderCRID:=List of Unified Service In-Parameter(List of pProviderCRID) DisconnectCR.ind(Interface Pointer, Count, List of pProviderCRID)	ACTIVE
405	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=DisconnectCR && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter DisconnectCR.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
406	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=DisconnectCR && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult DisconnectCR.cnf(-)(Interface Pointer, hresult)	ACTIVE
407	ACTIVE	SRTConnect.req(Interface Pointer, ProviderCRID, State, LastConnect, Count, List of pConnectIn) => Interface Pointer:=Interface Pointer Service Name:=SRTConnect List of Unified Service In-Parameter:=(ProviderCRID, State, LastConnect, Count, List of pConnectIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
408	ACTIVE	SRTConnect.rsp(+)(Interface Pointer, hresult, List of ppConnectOut) => Interface Pointer:=Interface Pointer Service Name:=SRTConnect hresult:=hresult List of Unified Service Out-Parameter:=List of ppConnectOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
409	ACTIVE	SRTConnect.rsp(-)(Interface Pointer, hresult) => Interface Pointer:=Interface Pointer Service Name:=SRTConnect hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
410	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=SRTConnect => Interface Pointer:=Interface Pointer ProviderCRID:=List of Unified Service In-Parameter(ProviderCRID) State:=List of Unified Service In-Parameter(State) LastConnect:=List of Unified Service In-Parameter(LastConnect) Count:=List of Unified Service In-Parameter(Count) List of pConnectIn:=List of Unified Service In-Parameter(List of pConnectIn) SRTConnect.ind(Interface Pointer, ProviderCRID, State, LastConnect, Count, List of pConnectIn)	ACTIVE
411	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=SRTConnect && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppConnectOut:=List of Unified Service Out-Parameter SRTConnect.cnf(+)(Interface Pointer, hresult, List of ppConnectOut)	ACTIVE
412	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=SRTConnect && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult SRTConnect.cnf(-)(Interface Pointer, hresult)	ACTIVE
413	ACTIVE	CoCreateInstance.req(Host address, Class ID, Interface ID) => Host address := Host address Class ID := Class ID Interface ID := Interface ID ARPM_CoCreateInstance.req(Host address, Class ID, Interface ID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
414	ACTIVE	CoCreateInstance.rsp(+)(hresult, Interface Pointer) => hresult:=hresult Interface Pointer:= Interface Pointer ARPM_CoCreateInstance.rsp(hresult, Interface Pointer)	ACTIVE
415	ACTIVE	CoCreateInstance.rsp(-)(hresult) => hresult:=hresult Interface Pointer:="empty" ARPM_CoCreateInstance.rsp(hresult, Interface Pointer)	ACTIVE
416	ACTIVE	ARPM_CoCreateInstance.ind(Host address, Class ID, Interface ID) => Host address := Host address Class ID := Class ID Interface ID := Interface ID CoCreateInstance.ind(Host address, Class ID, Interface ID)	ACTIVE
417	ACTIVE	ARPM_CoCreateInstance.cnf(hresult, Interface Pointer) /hresult.S=Success => Interface Pointer:= Interface Pointer hresult:=hresult CoCreateInstance.cnf(+)(hresult, Interface Pointer)	ACTIVE
418	ACTIVE	ARPM_CoCreateInstance.cnf(hresult, Interface Pointer) /hresult.S=Error => hresult:=hresult CoCreateInstance.cnf(-)(hresult)	ACTIVE
419	ACTIVE	CoDisconnectObject.req(Interface Pointer) => Interface Pointer:= Interface Pointer ARPM_CoDisconnectObject.req(Interface Pointer)	ACTIVE
420	ACTIVE	CoDisconnectObject.rsp(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer hresult:=hresult ARPM_CoDisconnectObject.rsp(Interface Pointer, hresult)	ACTIVE
421	ACTIVE	ARPM_CoDisconnectObject.ind(Interface Pointer) => Interface Pointer:= Interface Pointer CoDisconnectObject.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
422	ACTIVE	ARPM_CoDisconnectObject.cnf(Interface Pointer, hresult) => Interface Pointer:= Interface Pointer hresult:=hresult CoDisconnectObject.cnf(Interface Pointer, hresult)	ACTIVE
423	ACTIVE	Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter) => Interface Pointer:=Interface Pointer Service Name:=ARPM_Call List of Unified Service In-Parameter:=(Service Name, List of Unified Service In-Parameter) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
424	ACTIVE	Call.rsp(+)(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) => Interface Pointer:= Interface Pointer Service Name:=ARPM_Call hresult:=hresult List of Unified Service Out-Parameter:=Service Name, List of Unified Service Out-Parameter ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
425	ACTIVE	Call.rsp(-)(Interface Pointer, Service Name, hresult) => Interface Pointer:= Interface Pointer Service Name:=ARPM_Call hresult:=hresult List of Unified Service Out-Parameter:=Service Name ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
426	ACTIVE	ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter) /Service Name=ARPM_Call => Interface Pointer:=Interface Pointer Service Name=List of Unified Service In-Parameter(Service Name) List of Unified Service In-Parameter:=List of Unified Service In-Parameter(List of Unified Service In-Parameter) Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
427	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ARPM_Call && hresult.S=Success => Interface Pointer:=Interface Pointer Service Name=List of Unified Service Out-Parameter(Service Name) hresult:=hresult List of Unified Service Out-Parameter:=List of Unified Service In-Parameter(List of Unified Service Out-Parameter) Call.cnf(+)(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
428	ACTIVE	ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) /Service Name=ARPM_Call && hresult.S=Error => Interface Pointer:=Interface Pointer Service Name=List of Unified Service Out-Parameter hresult:=hresult Call.cnf(-)(Interface Pointer, Service Name, hresult)	ACTIVE

5.6 Application Relationship Protocol Machine (ARPM)

5.6.1 Overview

The type 10 specifies one AR endpoint class referred to as ORPC AR. Therefore, one instance of the state machine is specified and the service attribute AREP to address this state machine is omitted.

NOTE Additional ARPM machines can be added in future editions.

The class formal model is specified in IEC 61158-5-10.

5.6.2 Primitive definitions

5.6.2.1 Primitives Exchanged between FSPM and ARPM

The internal service primitives issued by FSPM to Application Relationship Protocol Machine (ARPM) are specified in Table 255.

Table 255 – Primitives issued by FSPM to ARPM

Primitive Names	Source	Associated Parameters	Functions
ARPM_Call.req	FSPM	Interface Pointer, Service Name, List of Unified Service In-Parameter	This primitive is used to request the ARPM to transfer a FAL service request.
ARPM_Call.rsp	FSPM	Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter	This primitive is used to request the ARPM to transfer a FAL service response.
ARPM_CoCreateInstance.req	FSPM	Host address, Class ID, Interface ID	This primitive is used to request the ARPM to transfer the FAL CoCreateInstance service request.
ARPM_CoCreateInstance.rsp	FSPM	hresult, Interface Pointer	This primitive is used to request the ARPM to transfer the FAL CoCreateInstance service response.
ARPM_CoDisconnectObject.req	FSPM	Interface Pointer	This primitive is used to request the ARPM to transfer the FAL CoDisconnectObject service request.
ARPM_CoDisconnectObject.rsp	FSPM	Interface Pointer, hresult	This primitive is used to request the ARPM to transfer the FAL CoDisconnectObject service response.

The internal service primitives issued by ARPM to FSPM are specified in Table 256.

Table 256 – Primitives issued by ARPM to FSPM

Primitive Names	Source	Associated Parameters	Functions
ARPM_Call.ind	ARPM	Interface Pointer, Service Name, List of Unified Service In-Parameter	This primitive is used to indicate the FSPM the reception of an ARPM Call.
ARPM_Call.cnf	ARPM	Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter	This primitive is used to indicate the FSPM the reception of an ARPM Call response.
ARPM_CoCreateInstance.ind	ARPM	Host address, Class ID, Interface ID	This primitive is used to indicate the FSPM the reception of an ARPM CoCreateInstance.
ARPM_CoCreateInstance.cnf	ARPM	hresult, Interface Pointer	This primitive is used to indicate the FSPM the reception of an ARPM CoCreateInstance response.
ARPM_CoDisconnectObject.ind	ARPM	Interface Pointer	This primitive is used to indicate the FSPM the reception of an ARPM CoDisconnectObject.
ARPM_CoDisconnectObject.cnf	ARPM	Interface Pointer, hresult	This primitive is used to indicate the FSPM the reception of an ARPM CoDisconnectObject response.

5.6.2.2 Parameters of FSPM/ARPM primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 257.

Table 257 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter Name	Description
Interface Pointer	This parameter specifies the interface address that uniquely represents the associated interface instance.
Service Name	This parameter specifies the service name of the FAL service.
List of Unified Service In-Parameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [in] attribute in one unified container.
List of Unified Service Out-Parameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [out] attribute in one unified container.
Host address	This parameter contains the IP address or DNS name of the node hosting the object to create.
Class ID	This parameter contains the Class ID of the object to create according to its FAL class specification and IDL specification.
Interface ID	This parameter contains the Interface ID of the interface of the created object. It is set according to its FAL class specification and IDL specification.
hresult	This parameter contains the result code.

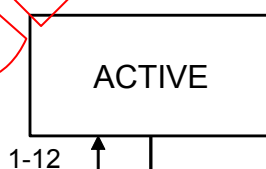
5.6.2.3 ARPM states

The defined state of the ARPM together with the description is listed in Table 258.

Table 258 – ARPM state descriptions

State Name	Description
ACTIVE	The ARPM in the ACTIVE state is ready to transmit and receive service primitives to and from the DMPM and the FSPM.

The state transition diagram of the ARPM is shown in Figure 52.

**Figure 52 – State transition diagram of ARPM**

5.6.2.4 ARPM state table

The ARPM state transitions are specified in Table 259.

Table 259 – ARPM state table

#	Current State	Event /Condition =>Action	Next State
1	ACTIVE	ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter) => pInterface:= Interface Pointer serviceName:=Service Name unifiedInParameter:=List of Unified Service In-Parameter referenceToIdl:=IDL Marshaling Reference DMPM_Call.req(pInterface, serviceName, unifiedInParameter, referenceToIdl)	ACTIVE
2	ACTIVE	ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter) => pInterface:= Interface Pointer serviceName:=Service Name hresult:=hresult unifiedOutParameter:=List of Unified Service Out-Parameter referenceToIdl:=IDL Marshaling Reference DMPM_Call.rsp(pInterface, serviceName, hresult, unifiedOutParameter, referenceToIdl)	ACTIVE
3	ACTIVE	ARPM_CoCreateInstance.req(Host address, Class ID, Interface ID) => hostAddress:=Host address classID:=Class ID interfaceID:=Interface ID DMPM_Create.req(hostAddress, classID, interfaceID)	ACTIVE
4	ACTIVE	ARPM_CoCreateInstance.rsp(, hresult, Interface Pointer) => hresult:=hresult pInterface:= Interface Pointer DMPM_Create.rsp(hresult, pInterface)	ACTIVE
5	ACTIVE	ARPM_CoDisconnectObject.req(Interface Pointer) => pInterface:= Interface Pointer DMPM_Disconnect.req(pInterface)	ACTIVE
6	ACTIVE	ARPM_CoDisconnectObject.rsp(Interface Pointer, hresult) => pInterface:= Interface Pointer hresult:=hresult DMPM_Disconnect.rsp(pInterface, hresult)	ACTIVE
7	ACTIVE	DMPM_Call.ind(pInterface, serviceName, unifiedInParameter) => Interface Pointer:=pInterface Service Name:=serviceName List of Unified Service In-Parameter:=unifiedInParameter ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
8	ACTIVE	DMPM_Call.cnf(pInterface, serviceName, hresult, unifiedOutParameter) => Interface Pointer:=pInterface Service Name:=serviceName hresult:=hresult List of Unified Service Out-Parameter:=unifiedOutParameter ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
9	ACTIVE	DMPM_Create.ind(hostAddress, classID, interfaceID) => Host address:=hostAddress Class ID:=classID Interface ID:=interfaceID ARPM_CoCreateInstance.ind(Host address, Class ID, Interface ID)	ACTIVE
10	ACTIVE	DMPM_Create.cnf(hresult, pInterface) => hresult:=hresult Interface Pointer:=pInterface ARPM_CoCreateInstance.cnf(hresult, Interface Pointer)	ACTIVE

11	ACTIVE	DMPM_Disconnect.ind(pInterface) => Interface Pointer:=pInterface ARPM_CoDisconnectObject.ind(Interface Pointer)	ACTIVE
12	ACTIVE	DMPM_Disconnect.cnf(pInterface, hresult) => Interface Pointer:=pInterface hresult:=hresult ARPM_CoDisconnectObject.cnf(Interface Pointer, hresult)	ACTIVE

5.7 DLL Mapping Protocol Machines

5.7.1 Overview

The DLL Mapping Protocol Machine (DMPM) mapping is represented by the ORPC (object oriented remote procedure call) model.

The ORPC model defines an abstract service interface that permits the underlying layer to exist at the data link layer, network layer, or transport layer. It permits, for example, using existing implementations of an object oriented remote procedure.

Additionally, the DLL mapping according 4.16.2 shall be applied for real-time transmission.

5.7.2 Primitive Definitions

5.7.2.1 Primitives Exchanged between DMPM and ARPM

The internal service primitives issued by ARPM to DMPM are specified in Table 260.

Table 260 – Primitives issued by ARPM to DMPM

Primitive Names	Source	Associated Parameters	Functions
DMPM_Call.req	ARPM	pInterface, serviceName, unifiedInParameter, referenceToId	This primitive is used to request the DMPM to transfer an ORPC call.
DMPM_Call.rsp	ARPM	pInterface, serviceName, hresult, unifiedOutParameter, referenceToId	This primitive is used to request the DMPM to transfer an ORPC response.
DMPM_Create.req	ARPM	hostAddress, classID, interfaceID	This primitive is used to request the DMPM to transfer an ORPC create.
DMPM_Create.rsp	ARPM	hresult, pInterface	This primitive is used to request the DMPM to transfer an ORPC create response.
DMPM_Disconnect.req	ARPM	pInterface	This primitive is used to request the DMPM to transfer an ORPC delete.
DMPM_Disconnect.rsp	ARPM	pInterface, hresult	This primitive is used to request the DMPM to transfer an ORPC delete response.

The internal service primitives issued by DMPM to ARPM are specified in Table 261.

Table 261 – Primitives issued by DMPM to ARPM

Primitive Names	Source	Associated Parameters	Functions
DMPM_Call.ind	DMPM	pInterface, serviceName, unifiedInParameter	This primitive is used to indicate the ARPM the transfer of an ORPC call.
DMPM_Call.cnf	DMPM	pInterface, serviceName, hresult, unifiedOutParameter	This primitive is used to indicate the ARPM the transfer of an ORPC response.
DMPM_Create.ind	DMPM	hostAddress, classID, interfaceID	This primitive is used to indicate the ARPM the transfer of an ORPC create.
DMPM_Create.cnf	DMPM	hresult, pInterface	This primitive is used to indicate the ARPM the transfer of an ORPC create response.
DMPM_Disconnect.ind	DMPM	pInterface	This primitive is used to indicate the ARPM the transfer of an ORPC delete.
DMPM_Disconnect.cnf	DMPM	pInterface, hresult	This primitive is used to indicate the ARPM the transfer of an ORPC delete response.

5.7.2.2 Parameters of ARPM/DMPM Primitives

The parameters used with the primitives exchanged between the ARPM and the DMPM are described in Table 262.

Table 262 – Parameters used with primitives exchanged between ARPM and DMPM

Parameter Name	Description
pInterface	This parameter specifies the interface address that uniquely represents the associated interface instance.
serviceName	This parameter specifies the service name of the FAL service.
unifiedInParameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [in] attribute in one unified container.
unifiedOutParameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [out] attribute in one unified container.
referenceToIdl	This parameter contains a reference to the IDL specification that shall be used by the abstract ORPC for data marshaling of the unified in and out parameter.
hostAddress	This parameter contains the IP address or DNS name of the node hosting the object to create.
classID	This parameter contains the Class ID of the object to create according to its FAL class specification and IDL specification.
interfaceID	This parameter contains the Interface ID of the object to create according to its FAL class specification and IDL specification.
hresult	This parameter contains the result code.

5.7.2.3 Primitives Exchanged between the ORPC model and DMPM

The internal service primitives issued by DMPM to ORPC model are specified in Table 263.

Table 263 – Primitives issued by DMPM to ORPC model

Primitive Names	Source	Associated Parameters	Functions
ORPC_Send.req	DMPM	plInterface, serviceName, unifiedInParameter, referenceToldl	This primitive is used to request the ORPC model to transfer an ORPC call.
ORPC_Send.rsp	DMPM	plInterface, serviceName, hresult, unifiedOutParameter, referenceToldl	This primitive is used to request the ORPC model to transfer an ORPC response.
ORPC_Create.req	DMPM	hostAddress, classID, interfaceID	This primitive is used to request the ORPC model to transfer an ORPC create.
ORPC_Create.rsp	DMPM	hresult, plInterface	This primitive is used to request the ORPC model to transfer an ORPC create response.
ORPC_Delete.req	DMPM	plInterface	This primitive is used to request the ORPC model to transfer an ORPC delete.
ORPC_Delete.rsp	DMPM	plInterface, hresult	This primitive is used to request the ORPC model to transfer an ORPC delete response.

The internal service primitives issued by the ORPC model to DMPM are specified in Table 264.

Table 264 – Primitives issued by ORPC model to DMPM

Primitive Names	Source	Associated Parameters	Functions
ORPC_Send.ind	ORPC model	plInterface, serviceName, unifiedInParameter	This primitive is used to indicate the DMPM the transfer of an ORPC call.
ORPC_Send.cnf	ORPC model	plInterface, serviceName, hresult, unifiedOutParameter	This primitive is used to indicate the DMPM the transfer of an ORPC response.
ORPC_Create.ind	ORPC model	hostAddress, classID, interfaceID	This primitive is used to indicate the DMPM the transfer of an ORPC create.
ORPC_Create.cnf	ORPC model	hresult, plInterface	This primitive is used to indicate the DMPM the transfer of an ORPC create response.
ORPC_Delete.ind	ORPC model	plInterface	This primitive is used to indicate the DMPM the transfer of an ORPC delete.
ORPC_delete.cnf	ORPC model	plInterface, hresult	This primitive is used to indicate the DMPM the transfer of an ORPC delete response.

5.7.2.4 Parameters of DMPM/ORPC model primitives

The parameters used with the primitives exchanged between the DMPM and the ORPC model are specified in Table 265.

Table 265 – Parameters used with primitives exchanged between DMPM and ORPC model

Parameter Name	Description
pInterface	This parameter specifies the interface address that uniquely represents the associated interface instance.
serviceName	This parameter specifies the service name of the FAL service.
unifiedInParameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [in] attribute in one unified container.
unifiedOutParameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [out] attribute in one unified container.
referenceToIdl	This parameter contains a reference to the IDL specification that shall be used by the abstract ORPC for data marshaling of the unified in and out parameter.
hostAddress	This parameter contains the IP address or DNS name of the node hosting the object to create.
classID	This parameter contains the Class ID of the object to create according to its FAL class specification and IDL specification.
interfaceID	This parameter contains the Interface ID of the object to create according to its FAL class specification and IDL specification.
hresult	This parameter contains the result code.

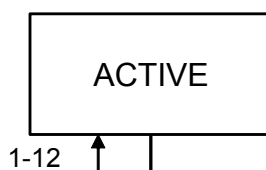
5.7.2.5 DMPM states

The defined state of the DMPM together with the description is listed in Table 266.

Table 266 – DMPM state descriptions

State Name	Description
ACTIVE	The DMPM in the ACTIVE state is ready to transmit and receive service primitives to and from the ORPC model and the ARPM.

The state transition diagram of the DMPM is shown in Figure 53.

**Figure 53 – State transition diagram of DMPM****5.7.2.6 DMPM state table**

The DMPM state transitions are specified in Table 267.

Table 267 – DMPM state table

#	Current State	Event /Condition =>Action	Next State
1	ACTIVE	DMPM_Call.req(pInterface, serviceName, unifiedInParameter, referenceToldI) => ORPC_Send.req(pInterface, serviceName, unifiedInParameter, referenceToldI)	ACTIVE
2	ACTIVE	DMPM_Call.rsp(pInterface, serviceName, hresult, unifiedOutParameter, referenceToldI) => ORPC_Send.rsp(pInterface, serviceName, hresult, unifiedOutParameter, referenceToldI)	ACTIVE
3	ACTIVE	DMPM_Create.req(hostAddress, classID, interfaceID) => ORPC_Create.req(hostAddress, classID, interfaceID)	ACTIVE
4	ACTIVE	DMPM_Create.rsp(hresult, pInterface) => ORPC_Create.rsp(hresult, pInterface)	ACTIVE
5	ACTIVE	DMPM_Disconnect.req(pInterface) => ORPC_Disconnect.req(pInterface)	ACTIVE
6	ACTIVE	DMPM_Disconnect.rsp(pInterface, hresult) => ORPC_Disconnect.rsp(pInterface, hresult)	ACTIVE
7	ACTIVE	ORPC_Send.ind(pInterface, serviceName, unifiedInParameter) => DMPM_Call.ind(pInterface, serviceName, unifiedInParameter)	ACTIVE
8	ACTIVE	ORPC_Send.cnf(pInterface, serviceName, hresult, unifiedOutParameter) => DMPM_Call.cnf(pInterface, serviceName, hresult, unifiedOutParameter)	ACTIVE
9	ACTIVE	ORPC_Create.ind(hostAddress, classID, interfaceID) => DMPM_Create.ind(hostAddress, classID, interfaceID)	ACTIVE
10	ACTIVE	ORPC_Create.cnf(hresult, pInterface) => DMPM_Create.cnf(hresult, pInterface)	ACTIVE
11	ACTIVE	ORPC_Disconnect.ind(pInterface) => DMPM_Disconnect.ind(pInterface)	ACTIVE
12	ACTIVE	ORPC_Disconnect.cnf(pInterface, hresult) => DMPM_Disconnect.cnf(pInterface, hresult)	ACTIVE

5.8 Protocol options

Type 10 has no protocol options.

6 Application layer protocol specification for decentralized periphery

6.1 FAL syntax description

6.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

6.1.2 APDU abstract syntax

Table 268 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 268 shall represent the content of the DLSDU in Table 4.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 268 – IO APDU substitutions

Substitution name	Structure
RTA-SDU	AlarmNotification-PDU ^ AlarmAck-PDU
BlockHeader	BlockType, BlockLength, BlockVersionHigh, BlockVersionLow
AlarmNotification-PDU	BlockHeader, AlarmType, API, SlotNumber, SubslotNumber, ModuleIdentNumber, SubmoduleIdentNumber, AlarmSpecifier, [MaintenanceItem], [AlarmItem] ^ [Upload&RetrievalItem] ^ [iParameterItem] The BlockType field within the BlockHeader shall be set to AlarmNotification high or low according Table 269.
AlarmAck-PDU	BlockHeader, AlarmType, API, SlotNumber, SubslotNumber, AlarmSpecifier, PNIOStatus The BlockType field within the BlockHeader shall be set to AlarmAck high or low according Table 269. 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)
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
AlarmItem	UserStructureIdentifier, Data* ^ ChannelDiagnosisData* ^ DiagnosisData* ^ ExtChannelDiagnosisData* ^ QualifiedChannelDiagnosisData* Special case ChannelDiagnosisData: UserStructureIdentifier(=0x8000), ChannelDiagnosisData* Special case Multiple for AlarmType Diagnosis: UserStructureIdentifier(=0x8001), DiagnosisData* Special case ExtChannelDiagnosisData: UserStructureIdentifier(=0x8002), ExtChannelDiagnosisData* Special case QualifiedChannelDiagnosisData: UserStructureIdentifier(=0x8003), QualifiedChannelDiagnosisData*
PROFINETIO-ServiceReqPDU	IODConnectReq ^ IODWriteReq ^ IODWriteMultipleReq ^ IODReadReq ^ IODControlReq ^ IODReleaseReq ^ IOXControlReq
PROFINETIO-ServiceResPDU	IODConnectRes ^ IODWriteRes ^ IODWriteMultipleRes ^ IODReadRes ^ IODControlRes ^ IODReleaseRes ^ IOXControlRes

Substitution name	Structure
IODConnectReq	ARBlockReq, {[IOCRBlockReq*], [AlarmCRBlockReq], [ExpectedSubmoduleBlockReq]^b, [PrmServerBlock], [MCRBlockReq*]^a, [ARRPCBlockReq] } Special case: "IOCAR" 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	ARBlockRes, {[IOCRBlockRes*], [AlarmCRBlockRes], [ModuleDiffBlock], [ARRPCBlockRes]^a } Special case: "IOCAR" 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, (IODWriteReqHeader, RecordDataWrite, [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
IODWriteMultipleRes	IODWriteResHeader^a, (IODWriteResHeader)* ^a The value of the parameter index shall be 0xE040.

Substitution name	Structure
IODWriteResHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* ^a , Index, RecordDataLength, AdditionalValue1, AdditionalValue2, PNIOStatus ^b , RWPadding* ^c ^a The number of padding octets (value=0) in the write response is 2. ^b In case of IODWriteRes PNIOStatus (=0,0,0,0). ^c The number of padding octets (value=0) in the write response is 16.
IODReadReq	IODReadReqHeader, [RecordDataReadQuery]
IODReadReqHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* ^a , Index, RecordDataLength, [TargetARUUID] ^b , RWPadding* ^c ^a The number of padding octets (value=0) in the read request is 2. ^b The optional field TargetARUUID shall only be used in conjunction with the value 0 of ARUUID (Implicit AR). ^c The number of padding octets (value=0) in the read request is 24 or 8.
RecordDataReadQuery	BlockHeader, Data* ^a ^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 standard.
IODReadRes	IODReadResHeader, RecordDataRead
IODReadResHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* ^a , Index, RecordDataLength, AdditionalValue1, AdditionalValue2, RWPadding* ^b ^a The number of padding octets (value=0) in the read res is 2. ^b The number of padding octets (value=0) in the read res is 20.
IODControlReq	ControlBlockConnect ^ ControlBlockPlug
IODControlRes	ControlBlockConnect ^ ControlBlockPlug ^ NULL ^a ^a 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	(ControlBlockConnect, [ModuleDiffBlock]) ^ (ControlBlockPlug, [ModuleDiffBlock]) ^ ControlBlockRFC ^ ControlBlockRTC The field ModuleDiffBlock shall only be present if an Application Ready has to signal an error.
IOXControlRes	ControlBlockConnect ^ ControlBlockPlug ^ NULL ^a ^a In case of a negative response NULL shall be transmitted.
ARBlockReq	BlockHeader, ARType, ARUUID, SessionKey, CMInitiatorMacAdd, CMInitiatorObjectUUID, ARProperties, CMInitiatorActivityTimeoutFactor, InitiatorUDPRTPort ^a , StationNameLength, CMInitiatorStationName ^a If no RT_CLASS_UDP CR is used, the value for InitiatorUDPRTPort shall be set to 0x8892. The BlockType field within the BlockHeader shall be set to ARBlockReq according Table 269.

Substitution name	Structure
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 Table 269.
IOCRBlockReq	BlockHeader, IOCRTYPE, IOCReference, LT, IOCProperties, DataLength, FrameID ^a, SendClockFactor, ReductionRatio, Phase, Sequence, FrameSendOffset, WatchdogFactor, DataHoldFactor, IOCRTagHeader, IOCRMulticastMACAdd, NumberOfAPIs, (API, NumberOfIODataObjects, (SlotNumber, SubslotNumber, IODataObjectFrameOffset)*, NumberOfIOCS, (SlotNumber, SubslotNumber, IOCSFrameOffset)*)* The BlockType field within the BlockHeader shall be set to IOCRBlockReq according Table 269. ^a The field FrameID shall be don't care if the field IOCRTYPE contains the value Output CR. Special Case IOCRTYPE = MULTICAST_CONSUMER_CR The field FrameID shall contain a FrameID from the defined multicast range. The configuration tool shall guarantee that this FrameID is unique within the project context. Special Case IOCRTYPE = OUTPUT_CR and IOCProperties = RT_CLASS_3 The field FrameID shall contain the FrameID used in the correspondig PDIRData.
MCRBlockReq	BlockHeader, IOCReference, AddressResolutionProperties, MCITimeoutFactor, StationNameLength, ProviderStationName, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
IOCRBlockRes	BlockHeader, IOCRTYPE, IOCReference, FrameID The BlockType field within the BlockHeader shall be set to IOCRBlockRes according Table 269.
ExpectedSubmodule-BlockReq	BlockHeader, NumberOfAPIs, (API, SlotNumber ^a, ModuleIdentNumber, ModuleProperties, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber, SubmoduleProperties ^b, (DataDescription, SubmoduleDataLength, LengthIOPS, LengthIOCS)*))* ^a This parameter is constant for one ExpectedSubmoduleBlockReq because each contains the data for 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 Table 269.

Substitution name	Structure
ModuleDiffBlock	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber^a, ModuleState^b, NumberOfSubmodules, [(SubslotNumber, SubmoduleIdentNumber^a, SubmoduleState)*])*)* ^a Real identification data ^b If ModuleState=NO_MODUL 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 parametrized. The BlockType field within the BlockHeader shall be set to ModuleDiffBlock according Table 269.
AlarmCRBlockReq	BlockHeader, AlarmCRTType, LT, AlarmCRProperties, RTATimeoutFactor, RTARetries, LocalAlarmReference, MaxAlarmDataLength, AlarmCRTagHeaderHigh, AlarmCRTagHeaderLow The BlockType field within the BlockHeader shall be set to AlarmCRBlockReq according Table 269.
AlarmCRBlockRes	BlockHeader, AlarmCRTType, LocalAlarmReference, MaxAlarmDataLength The BlockType field within the BlockHeader shall be set to AlarmCRBlockRes according Table 269.
PrmServerBlock	BlockHeader, ParameterServerObjectUID, ParameterServerProperties, CMInitiatorActivityTimeoutFactor, StationNameLength, ParameterServerStationName The BlockType field within the BlockHeader shall be set to PrmServerBlock according Table 269.
ARRPCBlockReq	BlockHeader, InitiatorRPCServerPort
ARRPCBlockRes	BlockHeader, ResponderRPCServerPort
ControlBlockConnect	BlockHeader, Padding*^a, ARUUID, SessionKey, Padding*^a, ControlCommand, ControlBlockProperties ^a The number of Padding octets shall be 2. The BlockType field within the BlockHeader shall be set to IOBlockReq, IOBlockRes, IOXBlockReq and, IOXBlockRes according Table 269.
ControlBlockPlug	BlockHeader, Padding*^a, ARUUID, SessionKey, AlarmSequenceNumber, ControlCommand, ControlBlockProperties ^a The number of Padding octets shall be 2. The BlockType field within the BlockHeader shall be set to IOXBlockReq, IOXBlockRes according Table 269. The field AlarmSequenceNumber shall contain the value of the sub-field AlarmSpecifier.SequenceNumber of the corresponding AlarmNotification-PDU.
ControlBlockRFC	BlockHeader, Padding*^a, ARUUID, SessionKey, Padding*^a, ControlCommand, ControlBlockProperties ^a The number of Padding octets shall be 2. The BlockType field within the BlockHeader shall be set to ReadyForCompanionBlock according Table 269.

Substitution name	Structure
ControlBlockRTC	BlockHeader, Padding* ^a, ARUUID, SessionKey, Padding* ^a, ControlCommand, ControlBlockProperties ^a The number of Padding octets shall be 2. The BlockType field within the BlockHeader shall be set to ReadyForRTCLASS3Block according Table 269.
ReleaseBlock	BlockHeader, Padding* ^a, ARUUID, SessionKey, Padding* ^a, ControlCommand, ControlBlockProperties ^a The number of Padding octets shall be 2. The BlockType field within the BlockHeader shall be set to ReleaseBlock according Table 269.
RecordDataRead	DiagnosisData* ^ ExpectedIdentificationData* ^ RealIdentificationData* ^ SubstituteValue ^ RecordInputDataObjectElement ^ RecordOutputDataObjectElement ^ Data* ^ ARData ^ IMData ^ LogData ^ ModuleDiffBlock ^ APIData ^ PDPortDataAdjust* ^ PDPortDataCheck* ^ PDPortDataReal* ^ PDIRData ^ PDSyncData* ^ PDevData ^ IsochronousModeData* ^ PDInterfaceMrpDataAdjust* ^ PDInterfaceMrpDataCheck* ^ PDInterfaceMrpDataReal* ^ PDPortMrpDataAdjust ^ PDPortMrpDataReal ^ PDPortFODataReal ^ PDPortFODataAdjust ^ PDPortFODataCheck ^ PDRealData* ^ PDExpectedData* ^ PDNCDataCheck ^ I&M0FilterData ^ ARFSUDDataAdjust ^ PDInterfaceFSUDDataAdjust ^ PDInterfaceDataReal ^ NULL ^a ^a NULL shall be used if a requested well-known data record is empty (e.g. Diagnosis, ModuleDiffBlock, ARData, ...).
RecordDataWrite	SubstituteValue ^ Data* ^ IMDataWrite ^ PDPortDataAdjust ^ PDPortDataCheck ^ PDIRData ^ PDSyncData ^ IsochronousModeData ^ PDInterfaceMrpDataAdjust ^ PDInterfaceMrpDataCheck ^ PDPortMrpDataAdjust ^ PDPortFODataAdjust ^ PDPortFODataCheck ^ PDNCDataCheck ^ ARFSUDDataAdjust ^ PDInterfaceFSUDDataAdjust
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(=diagnosis). Else, the field ChannelProperties.Specifier shall be set to disappear.
ChannelDiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType
Manufacturer-SpecificDiagnosis	SlotNumber, SubslotNumber, ChannelNumber, ChannelProperties, UserStructureIdentifier, Data*

Substitution name	Structure
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(=diagnosis). 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(=diagnosis). Else, the field ChannelProperties.Specifier shall be set to disappear.
QualifiedChannel-DiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType, ExtChannelErrorType, ExtChannelAddValue, QualifiedChannelQualifier
ExpectedIdentification-Data with BlockVersionLow = 0	BlockHeader, NumberOfSlots, (SlotNumber, ModuleIdentNumber, NumberOfSubslots, (SubslotNumber, SubmoduleIdentNumber))*
ExpectedIdentification-Data with BlockVersionLow = 1	BlockHeader, NumberOfAPIs, (API, NumberOfSlots, (SlotNumber, ModuleIdentNumber, NumberOfSubslots, (SubslotNumber, SubmoduleIdentNumber))*)*
RealIdentificationData with BlockVersionLow = 0	BlockHeader, 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 , PDIRBeginEndData
	^a This block may be omitted, if no RT_CLASS_3 multicast filtering is required or the destination it's requested.
PDSyncData with BlockVersionLow = 2	BlockHeader, Padding, Padding, PTCPSubdomainID, 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.
PDevData	BlockHeader, Padding, Padding, [PDIRData], [PDSyncData*]
PDRealData	MultipleBlockHeader, { [PDPortDataReal] ^b , [PDInterfaceMrpDataReal], [PDPortMrpDataReal], [PDPortFODDataReal] ^a , [PDInterfaceDataReal] } ^c
	^a There shall be no FiberOpticManufacturerSpecific information ^b The fields Slotnumber and Subslotnumber shall be ignored ^c For the data of each sub module (e.g. interface or port) is an own MultipleBlockHeader necessary

Substitution name	Structure
PDExpectedData	MultipleBlockHeader, { [PDPortDataCheck]^a, [PDPortDataAdjust]^a, [PDInterfaceMrpDataAdjust], [PDInterfaceMrpDataCheck], [PDPortMrpDataAdjust], [PDPortFODataAdjust], [PDPortFODataCheck], [PDNCDataCheck], [PDInterfaceFSUDataAdjust] } ^a The fields Slotnumber and Subslotnumber shall be ignored
PDPortDataReal	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, LengthOwnPortID, OwnPortID, NumberOfPeers, [Padding*]^a, [LengthPeerPortID, PeerPortID, LengthPeerChassisID, PeerChassisID, [Padding*]^a, LineDelay, PeerMACAddress^b, [Padding*]^a]*, MAUType^c, [Padding*]^a, Reserved^d, 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 409 ^d The number of reserved octets shall be 4.
PDInterfaceDataReal	BlockHeader, LengthOwnChassisID, OwnChassisID, [Padding*]^a, MACAddressValue^b, [Padding*]^a, IPPParameterValue, [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 submodul PNIOStatus(=0xDE, 0x80, 0xB0, 0x00)
RecordOutputData-ObjectElement	BlockHeader, SubstituteActiveFlag, LengthIOCS, LengthIOPS, LengthData, DataItem^a, SubstituteValue Special case: Response for an input submodul PNIOStatus(=0xDE, 0x80, 0xB0, 0x00) ^a For this record DataItem shall be coded as IOCS, Data, IOPS
ARData	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

Substitution name	Structure
APIData	BlockHeader, NumberOfAPIs, API*
LogData	BlockHeader, ActualLocalTimeStamp, NumberOfLogEntries, (LocalTimeStamp, ARUUID, PNIOStatus, EntryDetail)*
CMinitiatorObjectUUID	PROFINETIOConstantValue, InstanceHigh, InstanceLow, DeviceIdentNumber
ParameterServer-ObjectUUID	PROFINETIOConstantValue, InstanceHigh, InstanceLow, DeviceIdentNumber
DeviceIdentNumber	DeviceIDHigh, DeviceIDLow, VendorIDHigh, VendorIDLow
IMDataWrite	[I&M1] ^ [I&M2] ^ [I&M3] ^ [I&M4]
IMData	I&M0 ^ [I&M1] ^ [I&M2] ^ [I&M3] ^ [I&M4]
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&M0FilterData	{I&M0FilterDataSubmodul ^a , [I&M0FilterDataModul] ^b , [I&M0FilterDataDevice] ^c } ^a Shall contain all submodules with discrete IMData ^b Shall, if exists, contain only the module reference ^c Shall, if exists, contain only the device reference
I&M0FilterData-Submodul	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber)*))*
I&M0FilterDataModul	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber)*))*
I&M0FilterDataDevice	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber)*))*
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, 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. For a RX port exists no orange period.
TXPhaseAssignment	PhaseAssignment
RXPhaseAssignment	PhaseAssignment
PDIRFrameData	BlockHeader, Padding, Padding, (FrameSendOffset, DataLength, ReductionRatio, Phase, FrameID, EtherType, RxPort, FrameDetails, TxPortGroup, [Padding*] ^a)* ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.

Substitution name	Structure
IsochronousModeData	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, ControllerApplicationCycleFactor, TimeDataCycle, TimeIOInput, TimeIOOutput, TimeIOInputValid, TimeIOOutputValid
PDPortDataAdjust	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, { [AdjustDomainBoundary], [AdjustMulticastBoundary], [AdjustMAUType ^ AdjustLinkState], [AdjustPeerToPeerBoundary], [AdjustDCPBoundary] }
PDPortDataCheck	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, { [CheckPeers], [CheckLineDelay], [CheckMAUType], [CheckLinkState], [CheckSyncDifference], [CheckMAUTypeDifference] }
AdjustDomainBoundary with BlockVersionLow = 1	BlockHeader, Padding, Padding, DomainBoundaryIngress, DomainBoundaryEgress, AdjustProperties, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
AdjustMulticastBoundary	BlockHeader, Padding, Padding, MulticastBoundary, AdjustProperties, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
AdjustMAUType	BlockHeader, Padding, Padding, MAUType, AdjustProperties
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.
CheckPeers	BlockHeader, NumberOfPeers, (LengthPeerPortID, PeerPortID, LengthPeerChassisID, PeerChassisID)*, [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
CheckLinkState	BlockHeader, LinkState
CheckSyncDifference	BlockHeader, CheckSyncMode
CheckMAUType-Difference	BlockHeader, MAUTypeMode
PDInterface-MrpDataAdjust	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, [Padding*] ^a , MRP_LengthDomainName, MRP_DomainName, [Padding*] ^a , { [(MrpManagerParams, [MrpRTModeManagerData]) ^ (MrpClientParams, [MrpRTModeClientData])] } ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDInterface-MrpDataReal with BlockVersionLow = 1	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, MRP_Version, MRP_LengthDomainName, MRP_DomainName, [Padding*] ^a , { [(MrpManagerParams, [MrpRTModeManagerData]) ^ (MrpClientParams, [MrpRTModeClientData])], [MrpRingStateData], [MrpRTStateData] }, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDInterface-MrpDataCheck	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Check
PDPortMrpDataAdjust	BlockHeader, Padding, Padding, MRP_DomainUUID
PDPortMrpDataReal	BlockHeader, Padding, Padding, 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

Substitution name	Structure
MrpRTMode-ManagerData	BlockHeader, MRRT_TSTNRmax, MRRT_TSTdefaultT, [Padding*] ^a , MRP_RTMode ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
MrpRTModeClientData	BlockHeader, [Padding*] ^a , MRP_RTMode ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
MrpRingStateData	BlockHeader, MRP_RingState
MrpRTStateData	BlockHeader, MRP_RTState
PDPorFODDataReal	BlockHeader, Padding, Padding, FiberOpticType, FiberOpticCableType, [FiberOpticManufacturerSpecific*]
FiberOpticManufacturerSpecific	BlockHeader, VendorIDHigh, VendorIDLow, VendorBlockType, Data*, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDPorFODDataAdjust	BlockHeader, Padding, Padding, FiberOpticType, FiberOpticCableType
PDPorFODDataCheck	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
PDInterface-FSUDDataAdjust	BlockHeader, [Padding*] ^a , { [FSHelloBlock], [FastStartUpBlock] } ^b ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned. ^b At least one optional block shall be existing.
ARFSUDDataAdjust	BlockHeader, [Padding*] ^a , { [FSParameterBlock], [FastStartUpBlock] } ^b ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned. ^b At least one optional block shall be existing.
FSHelloBlock	BlockHeader, [Padding*] ^a , FSHelloMode, FSHelloInterval, FSHelloRetry, FSHelloDelay, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
FSPParameterBlock	BlockHeader, [Padding*] ^a , FSPParameterMode, FSPParameterUUID, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.
FastStartUpBlock	BlockHeader, [Padding*] ^a , Data*, [Padding*] ^a ^a The number of padding octets shall be adapted to make the block Unsigned32 aligned.

6.2 Transfer syntax

6.2.1 Coding section related to BlockHeader specific fields

6.2.1.1 Coding of the field BlockType

This field shall be coded as data type Unsigned16 with the values according to Table 269. This field identifies the PDU or data block type.

Withdrawing
IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 269 – BlockType

Value (hexadecimal)	Meaning	Use
0x0000	Reserved	Not used
0x0001	AlarmNotification High	AlarmNotification-PDU
0x0002	AlarmNotification Low	AlarmNotification-PDU
0x0003 – 0x0007	Reserved	—
0x0008	IODWriteReqHeader	RecordDataWrite
0x0009	IODReadReqHeader	RecordDataRead
0x000A – 0x000F	DiagnosisData	RecordDataRead, AlarmNotification-PDU
0x0010	DiagnosisData	RecordDataRead, AlarmNotification-PDU
0x0011	Reserved	—
0x0012	ExpectedIdentificationData	RecordDataRead
0x0013	RealIdentificationData	RecordDataRead
0x0014	SubstituteValue	RecordDataRead, RecordDataWrite
0x0015	RecordInputDataObjectElement	RecordDataRead
0x0016	RecordOutputDataObjectElement	RecordDataRead
0x0017	Reserved	—
0x0018	ARData	RecordDataRead
0x0019	LogData	RecordDataRead
0x001A	APIData	RecordDataRead
0x001B – 0x001F	Reserved	—
0x0020	I&M0	RecordDataRead
0x0021	I&M1	RecordDataRead RecordDataWrite
0x0022	I&M2	RecordDataRead RecordDataWrite
0x0023	I&M3	RecordDataRead RecordDataWrite
0x0024	I&M4	RecordDataRead RecordDataWrite
0x0025 – 0x002F	I&M5 – 15 ^a ^a Reserved for additional identification and maintenance data	RecordDataRead RecordDataWrite
0x0030	I&M0FilterDataSubmodul	I&M0FilterData
0x0031	I&M0FilterDataModul	I&M0FilterData
0x0032	I&M0FilterDataDevice	I&M0FilterData
0x0033 – 0x0100	Reserved	—
0x0101	ARBlockReq	IODConnectReq
0x0102	IOCRBlockReq	IODConnectReq
0x0103	AlarmCRBlockReq	IODConnectReq
0x0104	ExpectedSubmoduleBlockReq	IODConnectReq
0x0105	PrmServerBlockReq	IODConnectReq
0x0106	MCRBlockReq	IODConnectReq

Value (hexadecimal)	Meaning	Use
0x0107	Reserved	—
0x0108 – 0x010F	Reserved	—
0x0110	IOBlockReq, shall only be used in conjunction with connection establishment phase	IODControlReq (Prm End.req)
0x0111	IOBlockReq, shall only be used in conjunction with a plug alarm event	IODControlReq (Prm End.req)
0x0112	IOXBlockReq, shall only be used in conjunction with connection establishment phase	IOXControlReq (Application Ready.req)
0x0113	IOXBlockReq, shall only be used in conjunction with a plug alarm event	IOControlReq, IOSControlReq (Application Ready.req)
0x0114	ReleaseBlockReq	IODReleaseReq
0x0115	ARRPCServerBlockReq	IODConnectReq
0x0116	IOXBlockReq, shall only be used in conjunction with connection establishment phase	IOXControlReq (Ready for Companion.req)
0x0117	IOXBlockReq, shall only be used in conjunction with connection establishment phase	IOXControlReq (Ready for RT_CLASS_3.req)
0x0118 – 0x01FF	Reserved	—
0x0200	PDPortDataCheck	RecordDataRead RecordDataWrite
0x0201	PDevData	RecordDataRead
0x0202	PDPortDataAdjust	RecordDataRead RecordDataWrite
0x0203	PDSyncData	RecordDataRead RecordDataWrite
0x0204	IsochronousModeData	RecordDataRead RecordDataWrite
0x0205	PDIRData	RecordDataRead RecordDataWrite
0x0206	PDIRGlobalData	RecordDataRead RecordDataWrite
0x0207	PDIRFrameData	RecordDataRead RecordDataWrite
0x0208	PDIRBeginEndData	RecordDataRead RecordDataWrite
0x0209	Sub block for adjusting DomainBoundary	PDPortDataAdjust
0x020A	Sub block for checking Peers	PDPortDataCheck
0x020B	Sub block for checking LineDelay	PDPortDataCheck
0x020C	Sub block for checking MAUType	PDPortDataCheck
0x020E	Sub block for adjusting MAUType	PDPortDataAdjust
0x020F	PDPortDataReal	RecordDataRead RecordDataWrite
0x0210	Sub block for adjusting MulticastBoundary	PDPortDataAdjust
0x0211	PDInterfaceMrpDataAdjust	RecordDataRead RecordDataWrite
0x0212	PDInterfaceMrpDataReal	RecordDataRead
0x0213	PDInterfaceMrpDataCheck	RecordDataRead RecordDataWrite

Value (hexadecimal)	Meaning	Use
0x0214	PDPortMrpDataAdjust	RecordDataRead RecordDataWrite
0x0215	PDPortMrpDataReal	RecordDataRead
0x0216	Sub block for media redundancy manager parameters	PDInterfaceMrpDataAdjust PDInterfaceMrpDataReal
0x0217	Sub block for media redundancy client parameters	PDInterfaceMrpDataAdjust PDInterfaceMrpDataReal
0x0218	MrpRTModeManagerData Sub block for media redundancy RT mode for manager	PDInterfaceMrpDataAdjust PDInterfaceMrpDataReal
0x0219	MrpRingStateData Sub block for media redundancy ring state data	PDInterfaceMrpDataReal
0x021A	MrpRTStateData Sub block for media redundancy RT ring state data	PDInterfaceMrpDataReal
0x021B	Sub block for adjusting LinkState	PDPortDataAdjust
0x021C	Sub block for checking LinkState	PDPortDataCheck
0x021D	MrpRTModeClientData Sub block for media redundancy RT mode for clients	PDInterfaceMrpDataAdjust PDInterfaceMrpDataReal
0x021E	Sub block for checking local and remote CableDelay detected by LLDP to discover a sync difference	PDPortDataCheck, CheckSyncDifference
0x021F	Sub block for checking local and remote MAUTypes detected by LLDP	PDPortDataCheck, CheckMAUTypeDifference
0x0220	PDPortFODataReal	RecordDataRead
0x0221	Sub block for reading real fiber optic manufacturerspecific data	RecordDataRead
0x0222	PDPortFODataAdjust	RecordDataRead RecordDataWrite
0x0223	PDPortFODataCheck	RecordDataRead RecordDataWrite
0x0224	Sub block for adjusting PeerToPeerBoundary	PDPortDataAdjust
0x0225	Sub block for adjusting DCPBoundary	PDPortDataAdjust
0x0226 – 0x22F	Reserved	—
0x0230	PDNCDataCheck	RecordDataRead RecordDataWrite
0x0231 – 0x23F	Reserved	—
0x0240	PDInterfaceDataReal	RecordDataRead
0x0241 – 0x03FF	Reserved	—
0x0400	MultipleBlockHeader	RecordDataRead
0x0401 – 0x04FF	Reserved	—
0x0500	RecordDataReadQuery	RecordDataRead
0x0501 – 0x05FF	Reserved	—
0x0600	FSHelloBlock	PDInterfaceFSUDataAdjust
0x0601	FSPParameterBlock	ARFSUDataAdjust
0x0602 – 0x607	Reserved for FastStartUp	RecordDataRead RecordDataWrite

Value (hexadecimal)	Meaning	Use
0x0608	PDInterfaceFSUDataAdjust	RecordDataRead RecordDataWrite
0x0609	ARFSUDataAdjust	RecordDataRead RecordDataWrite
0x060A – 0x60F	Reserved for FastStartUp	RecordDataRead RecordDataWrite
0x0610 – 0x0EFF	Reserved	—
0x0F00	Maintenanceltem	Alarmnotification-PDU
0x0F01	Upload selected records within Upload&RetrievalItem	Alarmnotification-PDU
0x0F02	iParameterItem	Alarmnotification-PDU
0x0F03	Retrieve selected records within Upload&RetrievalItem	Alarmnotification-PDU
0x0F04	Retrieve all stored records within Upload&RetrievalItem	Alarmnotification-PDU
0x0F05 – 0x6FFF	Reserved	—
0x7000 – 0x7FFF	Reserved for user specific data objects	—
0x8000	Reserved	—
0x8001	Alarm Ack High	AlarmAck-PDU
0x8002	Alarm Ack Low	AlarmAck-PDU
0x8008	IODWriteResHeader	RecordDataWrite
0x8009	IODReadResHeader	RecordDataRead
0x800A – 0x8100	Reserved	—
0x8101	ARBlockRes	IODConnectRes
0x8102	IOCRBlockRes	IODConnectRes
0x8103	AlarmCRBlockRes	IODConnectRes
0x8104	ModuleDiffBlock	IODConnectRes, RecordDataRead, IOCControlReq, IOSControlReq
0x8105	PrmServerBlockRes	IODConnectRes
0x8106 – 0x810F	Reserved	—
0x8110	IODBlockRes, shall only be used in conjunction with connection establishment phase	IODControlRes (Prm End.rsp)
0x8111	IODBlockRes, shall only be used in conjunction with a plug alarm event	IODControlRes (Prm End.rsp)
0x8112	IOXBlockRes, shall only be used in conjunction with connection establishment phase	IOXControlRes (Application Ready.rsp)
0x8113	IOXBlockRes, shall only be used in conjunction with a plug alarm event	IOCControlRes IOSControlRes (Application Ready.rsp)
0x8114	ReleaseBlockRes	IODReleaseRes
0x8115	ARRPCServerBlockRes	IODConnectRes
0x8116	IOXBlockRes, shall only be used in conjunction with connection establishment phase	IOXControlRes (Ready for Companion.rsp)
0x8117	IOXBlockRes, shall only be used in conjunction with connection establishment phase	IOXControlRes (Ready for RT_CLASS_3.rsp)

Value (hexadecimal)	Meaning	Use
Other	Reserved	—

6.2.1.2 Coding of the field BlockLength

This field shall be coded as data type Unsigned16. This field shall contain the number of octets without counting the fields BlockType and BlockLength.

6.2.1.3 Coding of the field BlockVersionHigh

This field shall be coded as data type Unsigned8 with the values according to Table 270.

Table 270 – BlockVersionHigh

Value (hexadecimal)	Meaning	Use
0x00	Reserved	—
0x01	Version 1	Indicates Version 1
0x02 – 0xFF	Reserved	—

6.2.1.4 Coding of the field BlockVersionLow

This field shall be coded as data type Unsigned8 with the values according to Table 271.

Table 271 – BlockVersionLow

Value (hexadecimal)	Meaning	Use
0x00	Version 0	Indicates version 0
0x01	Version 1	Indicates version 1
0x02 – 0xFF	Version 2 to version 255	—

6.2.2 Coding section related to RTA-SDU specific fields

6.2.2.1 Coding of the field AlarmType

This field shall be coded as data type Unsigned16 with the values according to Table 272.

Table 272 – AlarmType

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Diagnosis
0x0002	Process
0x0003	Pull ^a
0x0004	Plug
0x0005	Status
0x0006	Update
0x0007	Redundancy
0x0008	Controlled by supervisor
0x0009	Released
0x000A	Plug Wrong Submodule
0x000B	Return of Submodule
0x000C	Diagnosis disappears
0x000D	Multicast communication mismatch notification
0x000E	Port data change notification
0x000F	Sync data changed notification
0x0010	Isochronous mode problem notification
0x0011	Network component problem notification
0x0012	Time data changed notification
0x0013 – 0x001D	Reserved
0x001E	Upload and retrieval notification
0x001F	Pull module ^b
0x0020 – 0x007F	Manufacturer specific
0x0080 – 0x00FF	Reserved for profiles
0x0100 – 0xFFFF	Reserved
^a With ARProperties.PullModuleAlarmAllowed(=0) subslot number 0x0001 – 0x8FFF used as “Pull submodule” and subslot number 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.	

6.2.2.2 Coding of the field AlarmSpecifier

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 10: AlarmSpecifier.SequenceNumber

This field shall be incremented with every AlarmNotification. The allowed range is from 0 to 2 047. By means of the SequenceNumber the receiver detects duplications and reflects the value of the field in the AlarmAck.

Bit 11: AlarmSpecifier.ChannelDiagnosis

This field shall be coded with the values according to Table 273. For all other AlarmTypes this field shall be set to zero.

Table 273 – AlarmSpecifier.ChannelDiagnosis

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Means that the SubslotNumber contains no ChannelDiagnosis, ExtChannelDiagnosis or QualifiedChannelDiagnosis with Channel-Properties.Maintenance (=diagnosis) and Channel-Properties.Specifier (=appear)	Diagnosis, Redundancy, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification
0x01	Means that the SubslotNumber contains at least one ChannelDiagnosis, ExtChannelDiagnosis or QualifiedChannelDiagnosis with Channel-Properties.Maintenance (=diagnosis) and Channel-Properties.Specifier (=appear)	

Bit 12: AlarmSpecifier.ManufacturerSpecificDiagnosis

This field shall be coded with the values according to Table 274. For all other AlarmTypes this field shall be set to zero.

Table 274 – AlarmSpecifier.ManufacturerSpecificDiagnosis

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Means that the SubslotNumber contains no ManufacturerSpecificDiagnosis with Channel-Properties.Maintenance (=diagnosis) and Channel-Properties.Specifier (=appear or disappear)	Diagnosis, Redundancy, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification
0x01	Means that the SubslotNumber contains at least one ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance (=diagnosis) and ChannelProperties.Specifier (=appear or disappear)	

Bit 13: AlarmSpecifier.SubmoduleDiagnosisState

This field shall be coded with the values according to Table 275. For all other AlarmTypes this field shall be set to zero.

Table 275 – AlarmSpecifier.SubmoduleDiagnosisState

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Error free No DiagnosisData with ChannelProperties.Maintenance (=diagnosis) 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 (=diagnosis) and ChannelProperties.Specifier (=disappear) may be present.	Diagnosis, Redundancy, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification
0x01	At least one DiagnosisData with Channel-Properties.Maintenance (=diagnosis) and Channel-Properties.Specifier (=appear) exists at the submodule.	

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 276. For all other AlarmTypes this field shall be set to zero.

Table 276 – AlarmSpecifier.ARDiagnosisState

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Error free No DiagnosisData with ChannelProperties.Maintenance(=diagnosis) 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(=diagnosis) and ChannelProperties.Specifier(=disappear) may be present.	Diagnosis, Redundancy, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification
0x01	At least one DiagnosisData with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=appear) exists at the AR.	

6.2.3 Coding section related to common address fields

6.2.3.1 Coding of the field API

This field, containing an administrative number, shall be coded as data type Unsigned32 according to Table 277.

Table 277 – API

Value (hexadecimal)	Meaning of API
0	Default
0x00000001 – 0x0000FFFF	Used for profile definitions fitting for 6.2.4.83.8
0x00010000 – 0xFFFFFFFF	Used for profile definitions beyond 6.2.4.83.8

NOTE 1 API means Application Process Identifier

6.2.3.2 Coding of the field SlotNumber

This field shall be coded as data type Unsigned16 according to Table 278.

Table 278 – 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 – 0xFFFF	Reserved

6.2.3.3 Coding of the field SubslotNumber

This field shall be coded as data type Unsigned16 according to Table 279.

Table 279 – SubslotNumber

Value (hexadecimal)	Meaning of SubslotNumber	
0	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	Used for 16 interface submodules with up to 255 ports. There may be gaps.	
0x9000 – 0xFFFF	Reserved	

The subslotnumber shall be evaluated with 0x8IPP with I counting interfaces (P := 00 means interface itself) and P counting ports.

6.2.3.4 Coding of the field Index

6.2.3.4.1 General

This field shall be coded as data type Unsigned16. The range from zero to 0x7FFF shall be used to address user specific record data objects. The range from 0x8000 to 0xFFFF shall be used for protocol specific function or further protocol extensions. The range from 0x8000 to 0xFFFF is divided into four sections. It is defined for each section whether the fields ARUUID, SlotNumber and SubslotNumber shall be ignored or used. The optional field TargetARUUID shall only be present and evaluated in conjunction with the defined indices.

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.

6.2.3.4.2 Usage of the address parameters

The following expressions show the evaluating of the address parameters according to the index range.

Expression 1 (subslot specific)

The rules to evaluate the address parameter shall be applied:

a) RPCOperationNmb == implicit Read

1) ARUUID == NIL (implicit AR)

i) TargetARUUID == NIL

API, Slot Number, Subslot Number, Index shall be evaluated

ii) TargetARUUID != NIL

API, Slot Number, Subslot Number, Index shall be evaluated

2) ARUUID != NIL (explicit AR)

not allowed

b) RPCOperationNmb == explicit Read / Write

1) ARUUID == NIL (implicit AR)

not allowed

2) ARUUID != NIL (established AR)

API, Slot Number, Subslot Number, Index shall be evaluated

Expression 2 (slot specific)

The rules to evaluate the address parameter shall be applied:

- a) RPCOperationNmb == implicit Read
 - 1) ARUUUID == NIL (implicit AR)
 - i) TargetARUUUID == NIL
API, Slot Number, Index shall be evaluated
 - ii) TargetARUUUID != NIL
API, Slot Number, Index shall be evaluated
 - 2) ARUUUID != NIL (explicit AR)
not allowed
- b) RPCOperationNmb == explicit Read / Write
 - 1) ARUUUID == NIL (implicit AR)
not allowed
 - 2) ARUUUID != NIL (established AR)
API, Slot Number, Index shall be evaluated

Expression 3 (AR specific)

The rules to evaluate the address parameter shall be applied:

- a) RPCOperationNmb == implicit Read
 - 1) ARUUUID == NIL (implicit AR)
 - i) TargetARUUUID == NIL
not allowed
 - ii) TargetARUUUID != NIL
Index shall be evaluated
 - 2) ARUUUID != NIL (explicit AR)
not allowed
- b) RPCOperationNmb == explicit Read / Write
 - 1) ARUUUID == NIL (implicit AR)
not allowed
 - 2) ARUUUID != NIL (established AR)
Index shall be evaluated

Expression 4 (API specific)

The rules to evaluate the address parameter shall be applied:

- a) RPCOperationNmb == implicit Read
 - 1) ARUUUID == NIL (implicit AR)
API, Index shall be evaluated
 - 2) ARUUUID != NIL (explicit AR)
not allowed
- b) RPCOperationNmb == explicit Read / Write
 - 1) ARUUUID == NIL (implicit AR)

not allowed

2) ARUUID != NIL (established AR)

API, Index shall be evaluated

Expression 5 (device specific)

The rules to evaluate the address parameter shall be applied:

a) RPCOperationNmb == implicit Read

1) ARUUID == NIL (implicit AR)

Index shall be evaluated

2) ARUUID != NIL (explicit AR)

not allowed

b) RPCOperationNmb == explicit Read / Write

1) ARUUID == NIL (implicit AR)

not allowed

2) ARUUID != NIL (established AR)

Index shall be evaluated

6.2.3.4.3 Grouping of DiagnosisData for the diagnosis records

The diagnosis records contain different data dependent on the index. The grouping shall be according to Table 280. The identifier is used in Table 282, Table 283, Table 284, Table 285, and Table 286.

Table 280 – Grouping of DiagnosisData

Identifier	Meaning
Diagnosis in channel coding	ChannelDiagnosisData, ExtChannelDiagnosisData, and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=appear)
Diagnosis in all codings	ChannelDiagnosisData, ExtChannelDiagnosisData, QualifiedChannelDiagnosis, and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=appear)
Maintenance required in channel coding	ChannelDiagnosisData, ExtChannelDiagnosisData, and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=MaintenanceRequired) and ChannelProperties.Specifier(=appear)
Maintenance required in all codings	ChannelDiagnosisData, ExtChannelDiagnosisData, QualifiedChannelDiagnosis, and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=MaintenanceRequired) and ChannelProperties.Specifier(=appear)
Maintenance demanded in channel coding	ChannelDiagnosisData, ExtChannelDiagnosisData, and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=MaintenanceDemanded) and ChannelProperties.Specifier(=appear)
Maintenance demanded in all codings	ChannelDiagnosisData, ExtChannelDiagnosisData, QualifiedChannelDiagnosis, and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=MaintenanceDemanded) and ChannelProperties.Specifier(=appear)
Diagnosis, Maintenance, Qualified and Status	ChannelDiagnosisData, ExtChannelDiagnosisData, QualifiedChannelDiagnosis, ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=any) and ChannelProperties.Specifier(=appear), and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=disappear)

6.2.3.4.4 Assigned numbers

The coding of the field index for user specific record data shall be according to Table 281.

Table 281 – Index (user specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUID, API, SlotNumber, SubslotNumber and TargetARUID
0 – 0x7FFF	User specific RecordData	Expression 1 applies.

The coding for submodule specific defined record data shall be according to Table 282.

Withdrawing
IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 282 – Index (subslot specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUID, API, SlotNumber, SubslotNumber and TargetARUID
0x8000	ExpectedIdentificationData for one subslot	Expression 1 applies.
0x8001	RealIdentificationData for one subslot	Expression 1 applies.
0x8002 – 0x8009	Reserved	Reserved
0x800A	Diagnosis in channel coding for one subslot	Expression 1 applies.
0x800B	Diagnosis in all codings for one subslot	Expression 1 applies.
0x800C	Diagnosis, Maintenance, Qualified and Status for one subslot	Expression 1 applies.
0x800D – 0x800F	Reserved	Reserved
0x8010	Maintenance required in channel coding for one subslot	Expression 1 applies.
0x8011	Maintenance demanded in channel coding for one subslot	Expression 1 applies.
0x8012	Maintenance required in all codings for one subslot	Expression 1 applies.
0x8013	Maintenance demanded in all codings for one subslot	Expression 1 applies.
0x8014 – 0x801D	Reserved	Reserved
0x801E	SubstituteValues for one subslot	Expression 1 applies.
0x801F – 0x8027	Reserved	Reserved
0x8028	RecordInputDataObjectElement for one subslot	Expression 1 applies.
0x8029	RecordOutputDataObjectElement for one subslot	Expression 1 applies.
0x802A	PDPortDataReal for one subslot	Expression 1 applies.
0x802B	PDPortDataCheck for one subslot	Expression 1 applies.
0x802C	PDIRData for one subslot	Expression 1 applies.
0x802D	Expected PDSyncData for one subslot with SyncID value 0	Expression 1 applies.
0x802E	Reserved	Reserved
0x802F	PDPortDataAdjust for one subslot	Expression 1 applies.
0x8030	IsochronousModeData for one subslot	Expression 1 applies.
0x8031	Expected PDSyncData for one subslot with SyncID value 1	Expression 1 applies.
0x8032 – 0x804E	Expected PDSyncData for one subslot with SyncID value 2..30	Expression 1 applies.
0x804F	Expected PDSyncData for one subslot with SyncID value 31	Expression 1 applies.
0x8050	PDInterfaceMrpDataReal for one subslot	Expression 1 applies.
0x8051	PDInterfaceMrpDataCheck for one subslot	Expression 1 applies.
0x8052	PDInterfaceMrpDataAdjust for one subslot	Expression 1 applies.
0x8053	PDPortMrpDataAdjust for one subslot	Expression 1 applies.
0x8054	PDPortMrpDataReal for one subslot	Expression 1 applies.
0x8055 – 0x805F	Reserved	Reserved
0x8060	PDPortFODDataReal for one subslot	Expression 1 applies.
0x8061	PDPortFODDataCheck for one subslot	Expression 1 applies.
0x8062	PDPortFODDataAdjust for one subslot	Expression 1 applies.
0x8063 – 0x806F	Reserved	Reserved

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUIID, API, SlotNumber, SubslotNumber and TargetARUUIID
0x8070	PDNCDataCheck for one subslot	Expression 1 applies.
0x8071 – 0x807F	Reserved	Reserved
0x8080	PDInterfaceDataReal for one subslot	Expression 1 applies.
0x8081 – 0x808F	Reserved	Reserved
0x8090	Expected PDInterfaceFSUDataAdjust	Expression 1 applies.
0x8091 – 0xAFEF	Reserved	Reserved
0xAFF0	I&M0	Expression 1 applies.
0xAFF1	I&M1	Expression 1 applies.
0xAFF2	I&M2	Expression 1 applies.
0xAFF3	I&M3	Expression 1 applies.
0xAFF4	I&M4	Expression 1 applies.
0xAFF5 – 0xAFFF	I&M5 – I&M15 ^a ^a Reserved for additional identification and maintenance data	Expression 1 applies.
0xB000 – 0xBFFF	Reserved for profiles	Expression 1 applies.

The coding for module specific defined record data shall be according to Table 283.

Table 283 – Index (slot specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUIID, API, SlotNumber, SubslotNumber and TargetARUUIID
0xC000	ExpectedIdentificationData for one slot	Expression 2 applies.
0xC001	RealIdentificationData for one slot	Expression 2 applies.
0xC002 – 0xC009	Reserved	Reserved
0xC00A	Diagnosis in channel coding for one slot	Expression 2 applies.
0xC00B	Diagnosis in all codings for one slot	Expression 2 applies.
0xC00C	Diagnosis, Maintenance, Qualified and Status for one slot	Expression 2 applies.
0xC00D – 0xC00F	Reserved	Reserved
0xC010	Maintenance required in channel coding for one slot	Expression 2 applies.
0xC011	Maintenance demanded in channel coding for one slot	Expression 2 applies.
0xC012	Maintenance required in all codings for one slot	Expression 2 applies.
0xC013	Maintenance demanded in all codings for one slot	Expression 2 applies.
0xC014 – 0xCFFF	Reserved	Reserved
0xD000 – 0xDFFF	Reserved for profiles	Expression 2 applies.

The coding for AR specific defined record data shall be according to Table 284.

Table 284 – Index (AR specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUID, API, SlotNumber, SubslotNumber and TargetARUUID
0xE000	ExpectedIdentificationData for one AR	Expression 3 applies.
0xE001	RealIdentificationData for one AR	Expression 3 applies.
0xE002	ModuleDiffBlock for one AR	Expression 3 applies.
0xE003 – 0xE009	Reserved	Reserved
0xE00A	Diagnosis in channel coding for one AR	Expression 3 applies.
0xE00B	Diagnosis in all codings for one AR	Expression 3 applies.
0xE00C	Diagnosis, Maintenance, Qualified and Status for one AR	Expression 3 applies.
0xE00D – 0xE00F	Reserved	Reserved
0xE010	Maintenance required in channel coding for one AR	Expression 3 applies.
0xE011	Maintenance demanded in channel coding for one AR	Expression 3 applies.
0xE012	Maintenance required in all codings for one AR	Expression 3 applies.
0xE013	Maintenance demanded in all codings for one AR	Expression 3 applies.
0xE014 – 0xE02F	Reserved	Reserved
0xE030	Reserved	Reserved
0xE031 – 0xE03F	Reserved	Reserved
0xE040	MultipleWrite	Expression 3 applies.
0xE041 – 0xE04F	Reserved	Reserved
0xE050	ARFSUDataAdjust data for one AR	Expression 3 applies.
0xE051 – 0xE05F	Reserved for FastStartUp	Expression 3 applies.
0xE060 – 0xEBFF	Reserved	Reserved
0xEC00 – 0xFFFF	Reserved for profiles	Expression 3 applies.

The coding for API specific defined record data shall be according to Table 285 if no AR has been established.

Table 285 – Index (API specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUID, API, SlotNumber, SubslotNumber and TargetARUID
0xF000	RealIdentificationData for one API	Expression 4 applies.
0xF001 – 0xF009	Reserved	Reserved
0xF00A	Diagnosis in channel coding for one API	Expression 4 applies.
0xF00B	Diagnosis in all codings for one API	Expression 4 applies.
0xF00C	Diagnosis, Maintenance, Qualified and Status for one API	Expression 4 applies.
0xF00D – 0xF00F	Reserved	Reserved
0xF010	Maintenance required in channel coding for one API	Expression 4 applies.
0xF011	Maintenance demanded in channel coding for one API	Expression 4 applies.
0xF012	Maintenance required in all codings for one API	Expression 4 applies.
0xF013	Maintenance demanded in all codings for one API	Expression 4 applies.
0xF014 – 0xF01F	Reserved	Reserved
0xF020	ARData for one API	Expression 4 applies.
0xF021 – 0xF3FF	Reserved	Reserved
0xF400 – 0xF7FF	Reserved for profiles	Expression 4 applies.

The coding for device specific defined record data shall be according to Table 286.

Table 286 – Index (device specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUID, API, SlotNumber, SubslotNumber and TargetARUID
0xF800 – 0xF80B	Reserved	Reserved
0xF80C	Diagnosis, Maintenance, Qualified and Status for one device	Expression 5 applies.
0xF80D – 0xF81F	Reserved	Reserved
0xF820	ARData	Expression 5 applies.
0xF821	APIData	Expression 5 applies.
0xF822 – 0xF82F	Reserved	Reserved
0xF830	LogData	Expression 5 applies.
0xF831	PDevData	Expression 5 applies.
0xF832 – 0xF83F	Reserved	Reserved
0xF840	I&M0FilterData	Expression 5 applies.
0xF841	PDRealData	Expression 5 applies.
0xF842	PDExpectedData	Expression 5 applies.
0xF843 – 0xFBFF	Reserved	Reserved
0xFC00 – 0xFFFF	Reserved for profiles ^a	Expression 5 applies.
^a The index is an administrative number.		

6.2.4 Coding section related to AL services

6.2.4.1 Coding of the field RecordDataLength

This field shall be coded as data type Unsigned32. The field RecordDataLength shall only contain the number of user octets.

6.2.4.2 Coding of the field SeqNumber

This field shall be coded as data type Unsigned16.

6.2.4.3 Coding of the field ARTYPE

This field shall be coded as data type Unsigned16 with the values according to Table 287.

Table 287 – ARType

Value (hexadecimal)	Meaning	Use
0x0000	Reserved	—
0x0001	IOCARSingl	—
0x0002	Reserved	—
0x0003	IOCARCIR	For future versions
0x0004	IOCAR_IOControllerRedundant	For future versions
0x0005	IOCAR_IODeviceRedundant	For future versions
0x0006	IOSAR	The supervisor AR is a special form of the IOCARSingl
0x0007 – 0x000F	Reserved	—
0x0010	IOCARSingl using RT_CLASS_3	—
0x0011 – 0xFFFF	Reserved	—

6.2.4.4 Coding of the field SessionKey

This field shall be coded as data type Unsigned16. The value of the field SessionKey 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 0 for the implicit AR.

6.2.4.5 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 IEEE 802 MAC address.

6.2.4.6 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 IEEE 802 MAC address, to the Table 288, and Table 289.

The IP Multicast address used for multicast communication relation using RT_CLASS_UDP shall be set according RFC2365.

Table 288 – 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 289 – IOCRMulticastMACAdd using RT_CLASS_2 or RT_CLASS_3

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 00-01-FF	Reserved for further multicast addresses within the Type 10 context
01-0E-CF	00-02-00 to 00-02-FF	RT_CLASS_2 multicast communication address
01-0E-CF	00-03-00 to 00-03-FF	Reserved for further multicast addresses within the Type 10 context
01-0E-CF	00-04-00 to 00-FF-FF	Reserved for other applications
01-0E-CF	01-00-00 to 01-FF-FF	—
01-0E-CF	02-00-00 to FF-FF-FF	Reserved for other applications

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

The IEEE Organizationally Unique Identifier for Type 10 is 00-0E-CF.

It shall be set according to the Table 290.

Table 290 – 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

6.2.4.7 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 IEEE 802 MAC address.

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

6.2.4.8 Coding of the field ARProperties

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 2: ARProperties.State

This field shall be set according to the Table 291.

Table 291 – ARProperties.State

Value (hexadecimal)	Meaning
0x00	Backup
0x01	Primary
0x02 – 0x07	Reserved

Bit 3: ARProperties.SupervisorTakeoverAllowed

This field shall be set according to Table 292.

Table 292 – ARProperties.SupervisorTakeoverAllowed

Value (hexadecimal)	Meaning
0x00	Not allowed
0x01	Allowed

Bit 4: ARProperties.ParametrizationServer

This field shall be set according to Table 293.

Table 293 – ARProperties.ParametrizationServer

Value (hexadecimal)	Meaning
0x00	External PrmServer
0x01	CM Initiator

Bit 5 – 7: ARProperties.reserved_1

This field shall be set according to 3.7.3.2.

Bit 8: ARProperties.DeviceAccess

This field shall be set according to Table 294.

Table 294 – 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 295.

Table 295 – 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 296.

Table 296 – 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 3.7.3.2.

Bit 24 – 30: ARProperties.reserved_3

This field shall be set to zero.

Bit 31: ARProperties.PullModuleAlarmAllowed

This field shall be set according to Table 297.

Table 297 – 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 – 0x8FFF shall code pulling of submodule in conjunction with AlarmType(=Pull).	Optional

6.2.4.9 Coding of the field IOCRProperties

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 3: IOCRProperties.RTClass

This field shall be set according to the Table 298.

Table 298 – IOCRProperties.RTClass

Value (hexadecimal)	Meaning	Use
0x00	Reserved	—
0x01	RT_CLASS_1	Data-RTC-PDU
0x02	RT_CLASS_2	Data-RTC-PDU
0x03	RT_CLASS_3	Data-RTC-PDU
0x04	RT_CLASS_UDP	UDP-RTC-PDU
0x05 – 0x07	Reserved	—

Bit 4 – 10: IOCRProperties.reserved_1

This field shall be set to zero.

Bit 11: IOCRProperties.MediaRedundancy

This field shall be set according to the Table 299.

Table 299 – IOCRProperties.MediaRedundancy

Value (hexadecimal)	Meaning
0x00	No media redundant frame transfer
0x01	Media redundant frame transfer

Bit 12 – 23: IOCRProperties.reserved_2

This field shall be set according to 3.7.3.2.

Bit 24 – 31: IOCRProperties.reserved_3

This field shall be set to zero.

6.2.4.10 Coding of the field NumberOfIODataObjects

This field shall be coded as data type Unsigned16.

6.2.4.11 Coding of the field NumberOfIOCS

This field shall be coded as data type Unsigned16.

6.2.4.12 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.

6.2.4.13 Coding of the field IOCRTagHeader

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0-11: IOCRTagHeader.IOCRVLANID

This field shall be set according to the Table 300.

Table 300 – IOCRTagHeader.IOCRVLANID

Value (hexadecimal)	Meaning
0x000	No VLAN
0x001	Default VLAN
0x002 – 0xFFFF	According IEEE 802.1Q

Bit 12: IOCRTagHeader.reserved

This field shall be set to zero.

Bit 13 – 15: IOCRTagHeader.IOUserPriority

This field shall be set according to the Table 301.

Table 301 – IOCRTagHeader.IOUserPriority

Value (hexadecimal)	Meaning
0x00 – 0x05	Reserved
0x06	IO CR Priority
0x07	Reserved

6.2.4.14 Coding of the field IOCRTYPE

This field shall be coded as data type Unsigned16 with the values according to Table 302.

Table 302 – IOCRTYPE

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Input CR
0x0002	Output CR
0x0003	Multicast Provider CR
0x0004	Multicast Consumer CR
0x0005 – 0xFFFF	Reserved

6.2.4.15 Coding of the field CMInitiatorActivityTimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 303 and Table 304.

Table 303 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess:=0

Value (decimal)	Meaning	Use
0	Reserved	—
1 – 1 000	With a time base of 100 ms	The IO device monitors the time between Connect response and the subsequent first activity of the IO controller
1 001 – 65 535	Reserved	—

Table 304 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess:=1

Value (decimal)	Meaning	Use
0 – 99	Reserved	—
100 – 1 000	With a time base of 100 ms	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	—

6.2.4.16 Coding of the field StationNameLength

This field shall be coded as data type Unsigned16.

6.2.4.17 Coding of the field CMInitiatorStationName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.15.1.

NOTE The field CMInitiatorStationName is not terminated by zero.

6.2.4.18 Coding of the field ParameterServerStationName

This field shall be coded as in 6.2.4.17.

6.2.4.19 Coding of the field ProviderStationName

This field shall be coded as in 6.2.4.17.

6.2.4.20 Coding of the field IODataObjectFrameOffset

This field shall be coded as data type Unsigned16. It is used within the IOCRBlockReq to reference the offset of the DataItem. It shall be in a range off 0 to 1 439. The maximum number of used octets shall not exceed the maximum C_SDU size.

6.2.4.21 Coding of the field IOCSFrameOffset

This field shall be coded as data type Unsigned16. It is used within the IOCRBlockReq to reference the offset of the IOCS. It shall be in a range off 0 to 1 439. The maximum number of used octets shall not exceed the maximum C_SDU size.

6.2.4.22 Coding of the field LengthIOCS

This field shall be coded as data type Unsigned8 and shall be set according to the Table 305.

Table 305 – LengthIOCS

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	One octet for IOCS
0x02 – 0xFF	Reserved

6.2.4.23 Coding of the field LengthIOPS

This field shall be coded as data type Unsigned8 and shall be set according to the Table 306.

Table 306 – LengthIOPS

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	One octet for IOPS
0x02 – 0xFF	Reserved

6.2.4.24 Coding of the field LengthData

This field shall be coded as data type Unsigned16. The values shall be in the range from 0 to 1 439.

6.2.4.25 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.

6.2.4.26 Coding of the field AlarmCRProperties

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0: AlarmCRProperties.Priority

This field shall be set according to the Table 307.

Table 307 – AlarmCRProperties.Priority

Value (hexadecimal)	Meaning
0x00	User priority (default), the priority given by the user is used and two alarm resources are available
0x01	Use only low priority, user priority is ignored and only one alarm resource is available

Bit 1: AlarmCRProperties.Transport

This field shall be set according to Table 308.

Table 308 – 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 3.7.3.2.

Bit 24 – 31: AlarmCRProperties.reserved_2

This field shall be set to zero.

6.2.4.27 Coding of the field AlarmCRTagHeaderHigh

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0-11: AlarmCRTagHeaderHigh.AlarmCRVLANID

This field shall be set according to the Table 309.

Table 309 – AlarmCRTagHeaderHigh.AlarmCRVLANID

Value (hexadecimal)	Meaning
0x000	No VLAN
0x001	Default VLAN
0x002 – 0xFFF	According IEEE 802.1Q

Bit 12: AlarmCRTagHeaderHigh.reserved

This field shall be set to zero.

Bit 13 – 15: AlarmCRTagHeaderHigh.AlarmUserPriority

This field shall be set according to the Table 310.

Table 310 – AlarmCRTagHeaderHigh.AlarmUserPriority

Value (hexadecimal)	Meaning
0x00 – 0x05	Reserved
0x06	Alarm CR Priority High
0x07	Reserved

6.2.4.28 Coding of the field AlarmCRTagHeaderLow

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0-11: AlarmCRTagHeaderLow.AlarmCRVLANID

This field shall be set according to the Table 311.

Table 311 – AlarmCRTagHeaderLow.AlarmCRVLANID

Value (hexadecimal)	Meaning
0x000	No VLAN
0x001	Default VLAN
0x002 – 0xFFF	According IEEE 802.1Q

Bit 12: AlarmCRTagHeaderLow.reserved

This field shall be set to zero.

Bit 13 – 15: AlarmCRTagHeaderLow.AlarmUserPriority

This field shall be set according to the Table 312.

Table 312 – AlarmCRTagHeaderLow.AlarmUserPriority

Value (hexadecimal)	Meaning
0x00 – 0x04	Reserved
0x05	Alarm CR Priority Low
0x06 – 0x07	Reserved

6.2.4.29 Coding of the field AlarmSequenceNumber

This field shall be coded as data type Unsigned16 with the values according to Table 313.

Table 313 – AlarmSequenceNumber

Value (hexadecimal)	Meaning	Use
0x000 – 0x7FF	Mirrors the sequence number of the plug alarm notification	It provides the relation between Plug Alarm notification and the ControlBlockPlug within Application Ready
0x800 – 0xFFFF	Reserved	—

6.2.4.30 Coding of the field AlarmCRTType

This field shall be coded as data type Unsigned16 with the values according to Table 314.

Table 314 – AlarmCRTType

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Alarm CR
0x0002 – 0xFFFF	Reserved

6.2.4.31 Coding of the field RTATimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 315. The time base is 100 ms. The RTATimeout is calculated:

$$\text{RTATimeout} = \text{RTATimeoutFactor} \times 100 \text{ ms} \quad (48)$$

Table 315 – RTATimeoutFactor

Value (hexadecimal)	Meaning	Use
0x0000	Reserved	—
0x0001 – 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.

6.2.4.32 Coding of the field RTARetries

This field shall be coded as data type Unsigned16 with the values according to Table 316.

Table 316 – RTARetries

Value (hexadecimal)	Meaning
0x0000 – 0x0002	Reserved
0x0003 – 0x000F	Mandatory
0x0010 – 0xFFFF	Reserved

6.2.4.33 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 NDREMapPDU. 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.

6.2.4.34 Coding of the field AddressResolutionProperties

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 2: AddressResolutionProperties.Protocol

This field shall be set according to Table 317.

Table 317 – 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 3.7.3.2.

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 318. The time base is one second. The AddressResolutionInterval is calculated:

$$\text{AddressResolutionInterval} = \text{AddressResolutionProperties.Factor} \times 1 \text{ s} \quad (49)$$

Table 318 – AddressResolutionProperties.Factor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x0064	Mandatory
0x0065 – 0xFFFF	Optional

6.2.4.35 Coding of the field MCITimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 319. The time base shall be 100 ms. The MCIMonitoringInterval is calculated:

$$\text{MCIMonitoringInterval} = \text{MCITimeoutFactor} \times 100 \text{ ms} \quad (50)$$

Table 319 – MCITimeoutFactor

Value (hexadecimal)	Meaning
0x0000 – 0x0064	Mandatory
0x0065 – 0xFFFF	Reserved

6.2.4.36 Coding of fields related to Instance, DeviceID, VendorID**6.2.4.36.1 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.

6.2.4.36.2 Coding of the field InstanceLow

This field shall be coded as data type Unsigned8.

6.2.4.36.3 Coding of the field InstanceHigh

This field shall be coded as data type Unsigned8.

NOTE The value 0 for both InstanceHigh and InstanceLow is recommended for single instance devices.

6.2.4.36.4 Coding of the field DeviceIDLow

This field shall be coded as data type Unsigned8.

6.2.4.36.5 Coding of the field DeviceIDHigh

This field shall be coded as data type Unsigned8.

NOTE The value 0 for both DeviceIDHigh and DeviceIDLow is reserved.

6.2.4.36.6 Coding of the field VendorIDLow

This field shall be coded as data type Unsigned8.

6.2.4.36.7 Coding of the field VendorIDHigh

This field shall be coded as data type Unsigned8. The value 0xFF is reserved for profiles.

NOTE The value 0 for both VendorIDHigh and VendorIDLow is reserved for IEC 61158–6–3 devices.

6.2.4.37 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 320.

Table 320 – ModuleIdentNumber

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0xFFFFFFFF	Manufacturer specific

6.2.4.38 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 321.

Table 321 – SubmoduleIdentNumber

Value (hexadecimal)	Meaning ARProperties.PullModuleAlarm- Allowed := 0	Meaning ARProperties.PullModuleAlarm- Allowed := 1
0x00000000	Assigned to the subslot zero only in conjunction with Pull alarm and Subslotnumber := 0 as identification for the function pull module. Manufacturer specific for subslots > 0	Manufacturer specific for all other subslots
0x00000001 – 0xFFFFFFFF	Manufacturer specific for all other subslots	

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, e.g. data structure, diagnosis and parametrization, defined in this standard are identified.

NOTE 2 The hardware of a submodule is identified by the VendorID and the OrderID.

6.2.4.39 Coding of the field NumberOfARs

This field shall be coded as data type Unsigned16.

6.2.4.40 Coding of the field NumberOfLOCKs

This field shall be coded as data type Unsigned16.

6.2.4.41 Coding of the field SubmoduleDataLength

This field shall be coded as data type Unsigned16. The range shall be between 0 and 1 439.

6.2.4.42 Coding of the field NumberOfModules

This field shall be coded as data type Unsigned16.

6.2.4.43 Coding of the field NumberOfSlots

This field shall be coded as data type Unsigned16.

6.2.4.44 Coding of the field NumberOfSubmodules

This field shall be coded as data type Unsigned16.

6.2.4.45 Coding of the field NumberOfSubslots

This field shall be coded as data type Unsigned16.

6.2.4.46 Coding of the field ARUUID

This field shall be coded as data type UUID. The value NIL indicates the usage of the implicit AR. All other values identify established ARs.

6.2.4.47 Coding of the field TargetARUUID

This field shall be coded as data type UUID. The value NIL indicates the usage of the implicit AR. All other values identify established ARs.

6.2.4.48 Coding of the field ActualLocalTimeStamp

This field shall be coded as data type Unsigned64. The value contains the current cycle count.

6.2.4.49 Coding of the field LocalTimeStamp

This field shall be coded as data type Unsigned64. The value contains the cycle count.

6.2.4.50 Coding of the field NumberOfLogEntries

This field shall be coded as data type Unsigned16. The value contains the number of log entries. The minimum number shall be 16 for an IO device and 4 Kocet an IO controller.

6.2.4.51 Coding of the field EntryDetail

This field shall be coded as data type Unsigned32 according to protocol machine behavior. The value zero shall be used for no detail.

6.2.4.52 Coding of the field AdditionalValue1 and AdditionalValue2

This field shall be coded as data type Unsigned16. The values shall contain additional user information within negative responses. The value zero indicates no further information.

6.2.4.53 Coding of the field ControlBlockProperties

The coding of this field shall be according to 3.7.3.4, Table 322 and Table 323.

Table 322 – ControlBlockProperties in conjunction with ControlCommand.ApplicationReady

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 3.7.3.2.

Table 323 – ControlBlockProperties in conjunction with the other values of the field ControlCommand

Bitposition	Value (hexadecimal)	Meaning
Bit 0 – 15	—	Shall be set according to 3.7.3.2.

6.2.4.54 Coding of the field ControlCommand

The coding of this field shall be according to 3.7.3.4. 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 the Table 324.

Table 324 – ControlCommand.PrmEnd

Value (hexadecimal)	Meaning	Use
0x00	no PrmEnd	—
0x01	The IO controller has finished the transmission of the stored start-up parameter.	IODControlReq

Bit 1: ControlCommand.ApplicationReady

This field shall be set according to the Table 325.

Table 325 – ControlCommand.ApplicationReady

Value (hexadecimal)	Meaning	Use
0x00	No Application Ready	—
0x01	The IO device has finished the start-up of its application or a new module or submodule was plugged and is ready to operate.	IOControlReq, IOSControlReq

Bit 2: ControlCommand.Release

This field shall be set according to the Table 326.

Table 326 – 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 the Table 327.

Table 327 – 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 the Table 328.

Table 328 – ControlCommand.ReadyForCompanion

Value (hexadecimal)	Meaning	Use
0x00	Not ready for companion	—
0x01	The IO device has finished the start-up of its application and is ready for the companion AR.	IOControlReq, IOSControlReq

Bit 5: ControlCommand.ReadyForRT_CLASS_3

This field shall be set according to the Table 329.

Table 329 – ControlCommand.ReadyForRT_CLASS_3

Value (hexadecimal)	Meaning	Use
0x00	Not ready for RT_CLASS_3	—
0x01	The IO device has finished the start-up of its application and is ready for RT_CLASS_3 communication.	IOControlReq, IOSControlReq

Bit 6 to 15: ControlCommand.reserved

This field shall be set to zero.

6.2.4.55 Coding of the field DataDescription

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: DataDescription.Type

This field shall be set according to the Table 330.

Table 330 – 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.

6.2.4.56 Coding of the field DataLength

This field shall be coded as data type Unsigned16. The values shall contain the length of the C_SDU. The range is from 40 to 1 440 for RT_CLASS_1 and RT_CLASS_2. The range for RT_CLASS_3 is from 0 to 1 440. The range for RT_CLASS_UDP is from 12 to 1 440.

Note The field DataLength is in minimum the sum of all DataItems. It can also be larger.

6.2.4.57 Coding of the field GAP

This field shall be coded as data type Unsigned8. The values shall be set to zero.

6.2.4.58 Coding of the field Padding

This field shall be coded as data type Unsigned8. The values shall be set to zero.

6.2.4.59 Coding of the field RTCPadding

This field shall be coded as data type Unsigned8.

6.2.4.60 Coding of the field RTAPadding

This field shall be coded as data type Unsigned8.

6.2.4.61 Coding of the field RWPadding

This field shall be coded as data type Unsigned8. The values shall be set to zero.

6.2.4.62 Coding of the field SendClockFactor

These fields shall be coded as data type Unsigned16 according to Table 331. The time base is 31,25 µs.

Table 331 – Values of SendClockFactor

Value (decimal)	Meaning	LengthOf-Period	Use
0	—	—	Reserved
1 – 7	Optional	—	—
8	Optional	250 µs	—
9 – 15	Optional	—	—
16	Optional	500 µs	—
17 – 31	Optional	—	—
32	Mandatory	1 ms	For valid combinations with ReductionRatio see Table 332
33 – 63	Optional	—	—
64	Optional	2 ms	—
65 – 127	Optional	—	—
128	Optional	4 ms	—
129 – 65 535	—	—	Reserved

NOTE 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 Equation (51).

$$\text{LengthOfPeriod} = \text{SendClockFactor} \times 31,25 \mu\text{s} \quad (51)$$

Used within the PDSyncData block, this field shall only be valid if the field SyncProperties.SyncID contains the value 0.

6.2.4.63 Coding of the field ReductionRatio

These fields shall be coded as data type Unsigned16 according to Table 332. The value range is from 1 to 16 384 in discrete steps.

Table 332 – Values of ReductionRatio

Value (decimal)	Meaning		Use
1	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
2	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
4	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
8	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
16	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
32	Mandatory ^a	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
64	Mandatory ^a	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
128	Mandatory ^a	RT_CLASS_x, RT_CLASS_UDP	With all SendClockFactor values
256	Mandatory ^b	RT_CLASS_x, RT_CLASS_UDP	For RT_CLASS_UDP with all SendClockFactor values
512	Mandatory ^c	RT_CLASS_x, RT_CLASS_UDP	For RT_CLASS_UDP with all SendClockFactor values
1 024	Mandatory	RT_CLASS_UDP	With all SendClockFactor values
2 048	Mandatory	RT_CLASS_UDP	With all SendClockFactor values
4 096	Mandatory	RT_CLASS_UDP	With all SendClockFactor values
8 192	Mandatory	RT_CLASS_UDP	Only with SendClockFactor values less or equal 64
16 384	Mandatory	RT_CLASS_UDP	Only with SendClockFactor values less or equal 32
16 385 – 65 535	Reserved	—	—
other	Optional	—	—

^a Optional for RT_CLASS_3 and for RT_CLASS_x with SendClockFactors not equal {1, 2, 4, 8, 16, 32, 64, 128}

^b Optional for RT_CLASS_3 and for RT_CLASS_x with SendClockFactors not equal {1, 2, 4, 8, 16, 32, 64}

^c Optional for RT_CLASS_3 and for RT_CLASS_x with SendClockFactors not equal {1, 2, 4, 8, 16, 32}

6.2.4.64 Coding of the field Phase

These fields shall be coded as data type Unsigned16 according to Table 333. It shall be less or equal to the related ReductionRatio.

Table 333 – 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	—

6.2.4.65 Coding of the field Sequence

These fields shall be coded as data type Unsigned16 according to Table 334.

Table 334 – Values of Sequence

Value (hexadecimal)	Meaning	Use
0	Mandatory, sequence undefined	Provider protocol machines defines the sequence for each phase
0x0001 – 0xFFFF	Optional, sequence defined	Provider protocol machines uses the given sequence for each phase

6.2.4.66 Coding of the field DataHoldFactor and WatchdogFactor

These fields shall be coded as data type Unsigned16. The time base is SendClockFactor multiplied with ReductionRatio of the monitored consumer. The WatchdogTime and the DataHoldTime are calculated.

$$\text{WatchdogTime} = \text{WatchdogFactor} \times \text{SendClockFactor} \times \text{ReductionRatio} \times 31,25 \mu\text{s} \quad (52)$$

$$\text{DataHoldTime} = \text{DataHoldFactor} \times \text{SendClockFactor} \times \text{ReductionRatio} \times 31,25 \mu\text{s} \quad (53)$$

The value range for Data-RTC-PDUs is from 0x0003 to 0x1E00. The DataHoldTime and WatchdogTime shall equal or less 1,92 s.

The value range for UDP-RTC-PDUs is from 0x0003 to 0xF000. The DataHoldTime and WatchdogTime shall equal or less 61,44 s.

For a frame, the usage of the DataHoldFactor is defined in Table 335 and of the WatchdogFactor is defined in Table 336.

Table 335 – DataHoldFactor of a frame

Value (hexadecimal)	Meaning	Description
0x0000	Reserved	—
0x0001 – 0x0002	Optional	An expiration of the time leads to an AR termination.
0x0003 – 0x00FF	Mandatory	An expiration of the time leads to an AR termination.
0x0100 – 0x1E00	Optional	An expiration of the time leads to an AR termination.
0x1E01 – 0xFFFF	Reserved	—

Table 336 – WatchdogFactor of a frame

Value (hexadecimal)	Meaning	Description
0x0000	Reserved	—
0x0001 – 0x0002	Optional	A value “1” leads to an AR termination for one missed frame.
0x0003 – 0x00FF	Mandatory	An expiration of the time leads to a check of the corresponding DataHoldFactor.
0x0100 – 0x1E00	Optional	An expiration of the time leads to a check of the corresponding DataHoldFactor.
0x1E01 – 0xFFFF	Reserved	—

6.2.4.67 Coding of the field FrameSendOffset

This field shall be coded as data type Unsigned32 and set according Table 337, Figure 54 and Equation (54). The time base is 1 ns.

NOTE The minimal planned gap between two RT_CLASS_3 frames is 1120 ns.

Table 337 – 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 Equation (54)	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

$$\text{FrameSendOffset}[\text{Frame}] + \text{Duration}[\text{Frame}] < \text{SendClockFactor} \times 31,25 \mu\text{s} \quad (54)$$

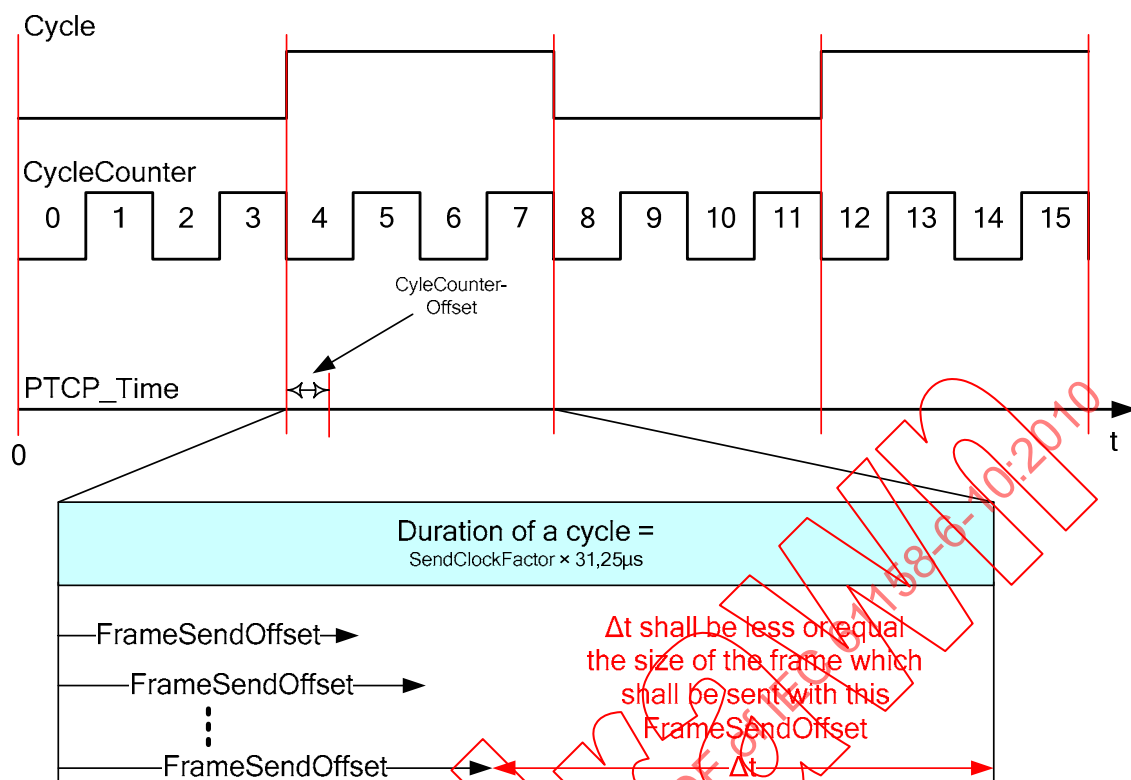


Figure 54 – FrameSendOffset vs. duration of a cycle

6.2.4.68 Coding section related to PNIOStatus

6.2.4.68.1 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=NULL and ARUID=NULL in ReadExpectedIdentification service.

6.2.4.68.2 Coding of the field ErrorCode

This field shall be coded as data type Unsigned8. The usage is defined in Table 338.

Table 338 – Values of ErrorCode for negative responses

Value (hexadecimal)	Meaning	Use
0x00 – 0x80	Reserved	—
0x81	PNIO	LogData
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, IOCControlRes, IOSControlRes	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	—

6.2.4.68.3 Coding of the field ErrorDecode

This field shall be coded as data type Unsigned8. The usage is defined in Table 339.

Table 339 – Values of ErrorDecode

Value (hexadecimal)	Meaning	Use
0x00 – 0x7F	Reserved	—
0x80	PNIORW ^a	Used in context with user error codes of the services Read, Write, Read Input Data, Read Output Data, Read Device Diagnosis, Read AR Data, Read Logbook, Read Expected Identification, Read Real Identification, ...
0x81	PNIO	Used in context with other services or internal e.g. RPC errors
0x82 – 0xFF	Reserved	—

^a Equivalent to IEC 61158-6-3 (DPV1)

6.2.4.68.4 Coding of the field ErrorCode1 and ErrorCode2

These fields shall be coded as data type Unsigned8.

The field ErrorDecode defines the coding and meaning of ErrorCode1 and ErrorCode2.

The ErrorDecode value PNIORW indicate that the parameters ErrorCode1 shall be set according Table 340. The ErrorCode1 consists of ErrorClass and ErrorDecode.

Table 340 – Coding of ErrorCode1 with ErrorDecode PNORW

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
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 PNORW indicates that the parameter ErrorCode2 shall be set user specific.

The ErrorDecode value PNIO indicates that the parameter ErrorCode1 shall be set according Table 341.

Table 341 – Values of ErrorCode1 and ErrorCode2 for ErrorDecode with the value PNIO

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
0	Reserved	0 – 255	Reserved
1	Connect Parameter Error Faulty ARBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		13	Error in Parameter NameOfStation
		14 – 255	Reserved
2	Connect Parameter Error Faulty IOCRBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		28	Error in Parameter IOCSFrameOffset production
		29 – 255	Reserved
3	Connect Parameter Error Faulty ExpectedSubmoduleBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		16	Error in Parameter LengthIOCS production
		17 – 255	Reserved
4	Connect Parameter Error Faulty AlarmCRBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		15	Error in Parameter AlarmCRTagHeaderLow
		16 – 255	Reserved
5	Connect Parameter Error Faulty PrmServerBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		8	Error in Parameter ParameterServerStationName
		9 – 255	Reserved
6	Connect Parameter Error Faulty MCRBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		8	Error in Parameter ProviderStationName
		9 – 255	Reserved
7	Connect Parameter Error Faulty ARRPCBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		4	Error in Parameter InitiatorRPCServerPort
		5 – 255	Reserved

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
8	Read Write Record Parameter Error Faulty Record	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		12	Error in Parameter TargetARUUID
		13 – 255	Reserved
9 – 19	Reserved	0 – 255	Reserved
20	IODControl Parameter Error Faulty ControlBlockConnect	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		9	Error in Parameter ControlBlockProperties
		10 – 255	Reserved
21	IODControl Parameter Error Faulty ControlBlockPlug	0	Error in Parameter BlockType
		...	...
		9	Error in Parameter ControlBlockProperties
		10 – 255	Reserved
22	IOXControl Parameter Error Faulty ControlBlock after a connection establishment	0	Error in Parameter BlockType
		...	...
		7	Error in Parameter ControlBlockProperties
		8 – 255	Reserved
23	IOXControl Parameter Error Faulty ControlBlock after a plug alarm	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		7	Error in Parameter ControlBlockProperties
		8 – 255	Reserved
24 – 39	Reserved	0 – 255	Reserved
40	Release Parameter Error Faulty ReleaseBlock	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		7	Error in Parameter ControlBlockProperties
		8 – 255	Reserved
41 – 49	Reserved	0 – 255	Reserved
50	Response Parameter Error Faulty ARBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		8	Error in Parameter ResponderUDPRTPort
		9 – 255	Reserved
51	Response Parameter Error Faulty IOCRBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
		6	Error in Parameter FrameID
		7 – 255	Reserved
52	Response Parameter Error Faulty AlarmCRBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		6	Error in Parameter MaxAlarmDataLengt
		7 – 255	Reserved
53	Response Parameter Error Faulty ModuleDiffBlock	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		13	Error in Parameter SubmoduleState
		14 – 255	Reserved
54	Response Parameter Error Faulty ARRPCBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		...	...
		4	Error in Parameter ResponderRPCServerPort
		5 – 255	Reserved
55 – 59	Reserved	0 – 255	Reserved
60	AlarmAck Error Codes	0	Alarm Type Not Supported
		1	Wrong Submodule State
		2 – 255	Reserved
61	CMDEV	0	State conflict
		1	Resource
		2 – 255	Usage see protocol machines and stored as Logbook entries
62	CMCTL	0	State conflict
		1	Timeout
		2	No data send
		3 – 255	Reserved
63	NRPM	0	No DCP active
		1	DNS Unknown_RealStationName
		2	DCP No_RealStationName
		3	DCP Multiple_RealStationName
		4	DCP No_StationName
		5	No_IP_Addr
		6	DCP_Set_Error
		7 – 255	Reserved
64	RMPM	0	ArgsLength invalid
		1	Unknown Blocks
		2	IOCR Missing
		3	Wrong AlarmCRBlock count

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
		4	Out of AR Resources
		5	AR UUID unknown
		6	State conflict
		7	Out of Provider, Consumer, or Alarm Resources
		8	Out of Memory
		9 – 255	Reserved
65	ALPMI	0	Invalid state
		1	Wrong ACK-PDU
		2 – 255	Reserved
66	ALPMR	0	Invalid state
		1	Wrong Notification PDU
		2 – 255	Reserved
67	LMPM	0 – 255	Usage see protocol machines and stored as Logbook entries
68	MMAC	0 – 255	Usage see protocol machines and stored as Logbook entries
69	RPC	0 – 255	See Table 342
70	APMR	0	Invalid state
		1	LMPM signaled an error
		2 – 255	Reserved
71	APMS	0	Invalid state
		1	LMPM signaled an error
		2	Timeout
		3 – 255	Reserved
72	CRM	0	Invalid state
		1 – 255	Reserved
73	PPM	0	Invalid state
		1 – 255	Reserved
74	Used by DCPUCS	0	Invalid state
		1	LMPM signaled an error
		2	Timeout
		3 – 255	Reserved
75	Used by DCPUCR	0	Invalid state
		1	LMPM signaled an error
		2 – 255	Reserved
76	Used by DCPMCS	0	Invalid state
		1	LMPM signaled an error
		2 – 255	Reserved
77	Used by DCPMCR	0	Invalid state
		1	LMPM signaled an error
		2 – 255	Reserved
78	FSPM	0 – 255	Usage see protocol machines and stored as Logbook entries

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
79 – 252	Reserved	0 – 255	Reserved
253	Used by RTA for protocol error (RTA_ERR_CLS_PROTOCOL)	0	Reserved
		1	Error within the coordination of sequence numbers (RTA_ERR_CODE_SEQ) error
		2	Instance closed (RTA_ERR_ABORT)
		3	AR out of memory (RTA_ERR_ABORT)
		4	AR add provider or consumer failed (RTA_ERR_ABORT)
		5	AR consumer DHT/WDT expired (RTA_ERR_ABORT)
		6	AR smi timeout (RTA_ERR_ABORT)
		7	AR alarm-open failed (RTA_ERR_ABORT)
		8	AR alarm-send.cnf(-) (RTA_ERR_ABORT)
		9	AR alarm-ack-send.cnf(-) (RTA_ERR_ABORT)
		10	AR alarm data too long (RTA_ERR_ABORT)
		11	AR alarm.ind(err) (RTA_ERR_ABORT)
		12	AR rpc-client call.cnf(-) (RTA_ERR_ABORT)
		13	AR abort.req (RTA_ERR_ABORT)
		14	AR re-run aborts existing (RTA_ERR_ABORT)
		15	AR release.ind received (RTA_ERR_ABORT)
		16	AR device deactivated (RTA_ERR_ABORT)
		17	AR removed (RTA_ERR_ABORT)
		18	AR protocol violation (RTA_ERR_ABORT)
		19	AR name resolution error (RTA_ERR_ABORT)
		20	AR RPC-Bind error (RTA_ERR_ABORT)

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
		21	AR RPC-Connect error (RTA_ERR_ABORT)
		22	AR RPC-Read error (RTA_ERR_ABORT)
		23	AR RPC-Write error (RTA_ERR_ABORT)
		24	AR RPC-Control error (RTA_ERR_ABORT)
		25	AR forbidden pull or plug after check.rsp and before in-data.ind (RTA_ERR_ABORT)
		26	AR AP removed (RTA_ERR_ABORT)
		27	AR link down (RTA_ERR_ABORT)
		28	AR could not register multicast-mac address (RTA_ERR_ABORT)
		29	Not synchronized (cannot start companion-ar) (RTA_ERR_ABORT)
		30	Wrong topology (cannot start companion-ar) (RTA_ERR_ABORT)
		31	DCP, station-name changed (RTA_ERR_ABORT)
		32	DCP, reset to factory-settings (RTA_ERR_ABORT)
		33	Can't start companion-AR because a 0x8ipp submodule in the first AR... (RTA_ERR_ABORT)
		34	No irdata record yet (RTA_ERR_ABORT)
		35	PDEV (RTA_ERROR_ABORT)
		36	PDEV, no port offers required speed/duplexity (RTA_ERROR_ABORT)
		37	IP-Suite [of the IOC] changed by means of DCP_set(IPParameter) or local engineering (RTA_ERROR_ABORT)
		38 – 200	Reserved
		201 – 255	Manufacturer specific
254	Reserved	0 – 255	Reserved
255	User specific	0 – 254	User specific

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
		255	Recommended for “User abort” without further detail

Table 342 – Values of ErrorCode2 for ErrorCode1 = RPC

Value (decimal)	Definition	Meaning
0	Reserved	—
1	CLRPC_ERR_REJECTED	Endpoint mapper or server did reject the call. For further details see Table 184 and Table 185.
2	CLRPC_ERR_FAULTED	Server had a fault while executing the call. For further details see Table 184 and Table 185.
3	CLRPC_ERR_TIMEOUT	Endpoint mapper or server did not respond
4	CLRPC_ERR_IN_ARGS	Broadcast or maybe “hdr_data” too large
5	CLRPC_ERR_OUT_ARGS	Server sent back more than “alloc_len”
6	CLRPC_ERR_DECODE	Result of endpoint mapper “lookup” could not be decoded
7	CLRPC_ERR_PNIO_OUT_ARGS	Out-args not “PNIO signature”, too short or inconsistent
8	CLRPC_ERR_PNIO_APP_TIMEOUT	RPC call was terminated after RPC application timeout
9 to 255	Reserved	—

6.2.4.69 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 343.

Table 343 – ModuleState

Value (hexadecimal)	Meaning	Use
0x0000	No module	E.g. module not plugged
0x0001	Wrong module ^a	E.g. ModuleIdentNumber wrong
0x0002	Proper module	Module is okay but at least one submodule is locked, wrong or missing
0x0003	Substitute	Module is not the same as requested – but the IO device was able to adapt by its own decision
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.		

6.2.4.70 Coding of the field SubmoduleState

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

6.2.4.70.1 Coding if SubmoduleState.FormatIndicator == 1**Bit 0 – 2: SubmoduleState.AddInfo**

This field shall be set according to the Table 344.

Table 344 – SubmoduleState.AddInfo

Value (hexadecimal)	Meaning	Description
0x00	None	—
0x01	Takeover is not allowed	This Submodul is not available for takeover by IOSAR.
0x02 – 0x07	Reserved	—

Bit 3: SubmoduleState.QualifiedInfo

This field shall be set according to the Table 345.

Table 345 – SubmoduleState.QualifiedInfo

Value (hexadecimal)	Meaning	Description
0x00	No QualifiedInfo available	No Channel of the Submodule contains QualifiedChannelDiagnosis
0x01	QualifiedInfo available	At least one Channel of the Submodule contains QualifiedChannelDiagnosis

Bit 4: SubmoduleState.MaintenanceRequired

This field shall be set according to the Table 346.

Table 346 – SubmoduleState.MaintenanceRequired

Value (hexadecimal)	Meaning	Description
0x00	No MaintenanceRequired available	No Channel of the Submodule requires maintenance
0x01	MaintenanceRequired available	At least one Channel of the Submodule requires maintenance

Bit 5: SubmoduleState.MaintenanceDemanded

This field shall be set according to the Table 347.

Table 347 – SubmoduleState.MaintenanceDemanded

Value (hexadecimal)	Meaning	Description
0x00	No MaintenanceDemanded available	No Channel of the Submodule demandes maintenance
0x01	MaintenanceDemanded available	At least one Channel of the Submodule demandes maintenance

Bit 6: SubmoduleState.DiagInfo

This field shall be set according to the Table 348.

Table 348 – SubmoduleState.DiagInfo

Value (hexadecimal)	Meaning	Description
0x00	No DiagnosisData available	There is no DiagnosisData available/stored for this submodule.
0x01	DiagnosisData available	There is DiagnosisData available for this submodule: It can be read with the corresponding records.

Bit 7 – 10: SubmoduleState.ARInfo

This field shall be set according to the Table 349.

Table 349 – 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. E.g. 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 an other IOAR
0x04	Locked By IO Supervisor (IOS)	This AR is not owner of the submodule. It is owned by an other IOSAR
0x05 – 0x0F	Reserved	—

Bit 11 – 14: SubmoduleState.IdentInfo

This field shall be set according to the Table 350.

Table 350 – SubmoduleState.IdentInfo

Value (hexadecimal)	Meaning
0x00	OK
0x01	Substitute (SU)
0x02	Wrong (WR)
0x03	NoSubmodule (NO)
0x04 – 0x0F	Reserved

Bit 15: SubmoduleState.FormatIndicator

This field shall be set according to the Table 351.

Table 351 – SubmoduleState.FormatIndicator

Value (hexadecimal)	Meaning	Use
0x00	Coding uses SubmoduleState.Detail	Shall 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.

6.2.4.70.2 Coding if SubmoduleState.FormatIndicator == 0**Bit 0 – 14: SubmoduleState.Detail**

This field shall be set according to the Table 352.

Table 352 – SubmoduleState.Detail

Value (hexadecimal)	Meaning	Use
0x0000	No submodule	—
0x0001	Wrong submodule	—
0x0002	Locked by IO controller	—
0x0003	Reserved	—
0x0004	Application ready pending	Shall only be used in conjunction with IOXControl request (application ready)
0x0005	Reserved	—
0x0006	Reserved	—
0x0007	Substitute	Submodule is not the same as requested – but compatible and the IO device was able to adapt In this case the IO device has to adapt (e.g. to new input or output length)
0x0008 – 0x7FFF	Reserved	—

NOTE The field SubmoduleState is only responded if there is a difference between expected and real configuration data. SubmoduleState “GOOD” is not defined because for such submodules no status is reported within the ExpectedSubmoduleRes.

Bit 15: SubmoduleState.FormatIndicator

This field shall be set according to the Table 351.

6.2.4.71 Coding of the field SubmoduleProperties

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: SubmoduleProperties.Type

This field shall be set according to the Table 353.

Table 353 – SubmoduleProperties.Type

Value (hexadecimal)	Meaning	Use
0x00	No input and no output data	Submodule without IO data, treated as input without data (for IOCS/IOPS), one Input DataDescription Block follows
0x01	Input data	Submodule with input data, one Input DataDescription Block follows
0x02	Output data	Submodule with output data, one Output DataDescription Block follows
0x03	Input and output data	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 354.

Table 354 – 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==0x02.

Bit 3: SubmoduleProperties.ReduceInputSubmoduleDataLength

This field shall be set according to Table 355.

Table 355 – 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

Bit 4: SubmoduleProperties.ReduceOutputSubmoduleDataLength

This field shall be set according to Table 356.

Table 356 – 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

Bit 5: SubmoduleProperties.DiscardIOXS

This field shall be set according to Table 357.

Table 357 – 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

Bit 6 – 7: SubmoduleProperties.reserved_1

This field shall be set according to 3.7.3.2.

Bit 8 – 15: SubmoduleProperties.reserved_2

This field shall be set to zero.

6.2.4.72 Coding of the field ModuleProperties

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: ModuleProperties.reserved_1

This field shall be set according to 3.7.3.2.

Bit 8 – 15: ModuleProperties.reserved_2

This field shall be set to zero.

6.2.4.73 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 358.

Table 358 – SubstitutionMode

Value (hexadecimal)	Meaning	Use
0x0000	ZERO	The outputs are set to zero or inactive
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	May be used in profile specification
0x0200 – 0xFFFF	Reserved	—

6.2.4.74 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 359.

Table 359 – 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	—

6.2.4.75 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 360.

Table 360 – InitiatorUDPRTPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Reserved
0x0400 – 0xBFFF	IANA_DEFINED_PORT	Usable
0x8892	Default for unicast IANA_PNIO_UDP_UNICAST_PORT	Recommended
0x8893	Default for multicast IANA_PNIO_UDP_MULTICAST_PORT	Recommended
0xC000 – 0xFFFF	IANA_FREE_PORT is used as dynamic and/or private port	Usable

6.2.4.76 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 361.

Table 361 – ResponderUDPRTPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	IANA_WELL_KNOWN_PORT	Reserved
0x0400 – 0xBFFF	IANA_DEFINED_PORT	Usable
0x8892	Default for unicast IANA_PNIO_UDP_UNICAST_PORT	Recommended
0x8893	Default for multicast IANA_PNIO_UDP_MULTICAST_PORT	Recommended
0xC000 – 0xFFFF	IANA_FREE_PORT is used as dynamic and/or private port	Usable

6.2.4.77 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 362.

Table 362 – 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

6.2.4.78 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 363.

Table 363 – 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

6.2.4.79 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. This field shall contain the value of the reference of the IO device within AlarmCRBlockRes and ARData.

6.2.4.80 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 within ARData.

6.2.4.81 Coding of the field MaxAlarmDataLength

This field shall be coded as data type Unsigned16. This field shall contain the maximal value of the AlarmNotification-PDU. The allowed values shall be in the range from 200 to 1 432.

6.2.4.82 Coding of the field ParameterServerProperties

This field shall be coded as data type Unsigned32. This field is reserved for future use.

6.2.4.83 Coding of the I&M Records**6.2.4.83.1 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.

6.2.4.83.2 Coding of the field IM_Serial_Number

This field shall be coded as data type VisibleString[16]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 16 octets.

6.2.4.83.3 Coding of the field IM_Hardware_Revision

This field shall be coded as data type Unsigned16 with the values according to Table 364.

Table 364 – IM_Hardware_Revision

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Hardware revision

6.2.4.83.4 Coding of the field IM_SWRevision_Functional_Enhancement

This field shall be coded as data type Unsigned8 with the values according to Table 365.

Table 365 – IM_SWRevision_Functional_Enhancement

Value (hexadecimal)	Meaning
0x00 – 0xFF	Functional Enhancement

6.2.4.83.5 Coding of the field IM_SWRevision_Bug_Fix

This field shall be coded as data type Unsigned8 with the values according to Table 366.

Table 366 – IM_SWRevision_Bug_Fix

Value (hexadecimal)	Meaning
0x00 – 0xFF	Bug Fix

6.2.4.83.6 Coding of the field IM_SWRevision_Internal_Change

This field shall be coded as data type Unsigned8 with the values according to Table 367.

Table 367 – IM_SWRevision_Internal_Change

Value (hexadecimal)	Meaning
0x00 – 0xFF	Internal Change

6.2.4.83.7 Coding of the field IM_Revision_Counter

This field shall be coded as data type Unsigned16 with the values according to Table 368.

Table 368 – IM_Revision_Counter

Value (hexadecimal)	Meaning	Usage
0x0000	Revision Counter	Mandatory
0x0001 – 0xFFFF	Revision Counter	Optional

6.2.4.83.8 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 369.

Table 369 – IM_Profile_ID

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Using shall be defined by profiles, number shall be uniquely administered.
0xFFFF	Using shall be defined by profiles, number shall be uniquely administered.

NOTE 1 Depends on 6.2.3.1

6.2.4.83.9 Coding of the field IM_Profile_Specific_Type

This field shall be coded as data type Unsigned16 with the values according to Table 370.

Table 370 – IM_Profile_Specific_Type

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Using shall be defined by profiles and will be treated as a subversion of the profile

6.2.4.83.10 Coding of the field IM_Version_Major

This field shall be coded as data type Unsigned8 with the values according to Table 371.

Table 371 – IM_Version_Major

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Shall set in this version
0x02 – 0xFF	Reserved

6.2.4.83.11 Coding of the field IM_Version_Minor

This field shall be coded as data type Unsigned8 with the values according to Table 372.

Table 372 – IM_Version_Minor

Value (hexadecimal)	Meaning
0x01	Shall set in this version
0x00, 0x02 – 0xFF	Reserved

6.2.4.83.12 Coding of the field IM_Supported

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0: IM_Supported.Profil_specific

This field shall be set related to the profiles.

Bit 1: IM_Supported.I&M1

This field shall be set if the I&M1 record contains data.

Bit 2: IM_Supported.I&M2

This field shall be set if the I&M2 record contains data.

Bit 3: IM_Supported.I&M3

This field shall be set if the I&M3 record contains data.

Bit 4: IM_Supported.I&M4

This field shall be set if the I&M4 record contains data.

Bit 5: IM_Supported.I&M5

This field shall be set if the I&M5 record contains data.

Bit 6: IM_Supported.I&M6

This field shall be set if the I&M6 record contains data.

Bit 7: IM_Supported.I&M7

This field shall be set if the I&M7 record contains data.

Bit 8: IM_Supported.I&M8

This field shall be set if the I&M8 record contains data.

Bit 9: IM_Supported.I&M9

This field shall be set if the I&M9 record contains data.

Bit 10: IM_Supported.I&M10

This field shall be set if the I&M10 record contains data.

Bit 11: IM_Supported.I&M11

This field shall be set if the I&M11 record contains data.

Bit 12: IM_Supported.I&M12

This field shall be set if the I&M12 record contains data.

Bit 13: IM_Supported.I&M13

This field shall be set if the I&M13 record contains data.

Bit 14: IM_Supported.I&M14

This field shall be set if the I&M14 record contains data.

Bit 15: IM_Supported.I&M15

This field shall be set if the I&M15 record contains data.

6.2.4.83.13 Coding of the field IM_Tag_Function

This field shall be coded as data type VisibleString[32]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 32 octets.

6.2.4.83.14 Coding of the field IM_Tag_Location

This field shall be coded as data type VisibleString[22]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 22 octets.

6.2.4.83.15 Coding of the field IM_Date

This field shall be coded as data type VisibleString[16] with the values according to Table 373. The string format “YYYY-MM-DD HH:MM” is in accordance with the ISO 8601.

NOTE The VisibleString[16] filled with the character blank ‘ ’ is the reset to factory value.

Table 373 – IM_Date

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

6.2.4.83.16 Coding of the field IM_Descriptor

This field shall be coded as data type VisibleString[54]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 54 octets.

NOTE The VisibleString[54] filled with the character blank ' ' is the reset to factory value.

6.2.4.83.17 Coding of the field IM_Signature

This field shall be coded as data type OctetString[54]. The value shall be set manufacturer specific and shall be filled with zero if it is shorter than 54 octets.

NOTE The OctetString[54] filled with zero or an application default is "reset to factory" value.

6.2.5 Coding section related to Alarm and Diagnosis PDUs

6.2.5.1 Coding of the field UserStructureIdentifier

This field shall be coded as data type Unsigned16 with the values according to Table 374. This field identifies the structure of the field Data of the AlarmNotification and structure of the field Data of the alarm data.

Withdrawing
IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 374 – UserStructureIdentifier

Value (hexadecimal)	Meaning	Usage
0x0000 – 0x7FFF	ManufacturerSpecific	ManufacturerSpecific Diagnosis shall be used in conjunction with following AlarmTypes: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification 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: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification ChannelDiagnosis shall be used in conjunction with following Records: All records containing Diagnosis Data
0x8001	Multiple	Shall only be used in conjunction with Diagnosis, Redundancy, DiagnosisDisappears, MulticastCommunicationMismatch, PortDataChangedNotification, SyncDataChangedNotification and IsochronousModeProblemNotification with data, which comply to the “(BlockHeader, Data)*” structure. Furthermore, the BlockType shall always correspond to the used AlarmType.
0x8002	ExtChannelDiagnosis	ExtChannelDiagnosis shall be used in conjunction with following AlarmTypes: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification ExtChannelDiagnosis shall be used in conjunction with following Records: All records containing Diagnosis Data
0x8003	QualifiedChannel-Diagnosis	QualifiedChannelDiagnosis shall be used in conjunction with following AlarmTypes: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification QualifiedChannelDiagnosis shall be used in conjunction with following Records: All records containing Diagnosis Data
0x8004 – 0x80FF	Reserved	—

Value (hexadecimal)	Meaning	Usage
0x8100	Maintenance	Maintenance shall be used in conjunction with following AlarmTypes: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification 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.
0x8101 – 0x81FF	Reserved	—
0x8200	Upload&Retrieval	Upload&Retrieval shall be used in conjunction with upload and retrieval notification.
0x8201	iParameter	iParameter shall be used in conjunction with upload and retrieval notification.
0x8202 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	—
0xA000 – 0xFFFF	Reserved	—

6.2.5.2 Coding of the field ChannelErrorType

This field shall be coded as data type Unsigned16 with the values according to Table 375.

Table 375 – ChannelErrorType

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	Manufacturer specific Recommended for “parameter missing”	The channel needs (additional) parameters. No or to less parameters are written
0x0010	Manufacturer specific Recommended for “parametrization fault”	Parametrization fault. Wrong or to many parameters are written
0x0011	Manufacturer specific Recommended for “power supply fault”	Power supply fault
0x0012	Manufacturer specific Recommended for “fuse blown / open”	Fuse blown / open
0x0013	Manufacturer specific Recommended for “communication fault”	Communication fault. Sequence number wrong / sequence wrong
0x0014	Manufacturer specific Recommended for “ground fault”	Ground fault
0x0015	Manufacturer specific Recommended for “reference point lost”	Reference point lost
0x0016	Manufacturer specific Recommended for “process event lost / sampling error”	Process event lost / sampling error
0x0017	Manufacturer specific Recommended for “threshold warning”	Threshold warning
0x0018	Manufacturer specific Recommended for “output disabled”	Output disabled
0x0019	Manufacturer specific Recommended for “safety event”	Safety event
0x001A	Manufacturer specific Recommended for “external fault”	External fault
0x001B	Manufacturer specific	Manufacturer specific
0x001C	Manufacturer specific	Manufacturer specific

Value (hexadecimal)	Meaning	Assigned text
0x001D	Manufacturer specific	Manufacturer specific
0x001E	Manufacturer specific	Manufacturer specific
0x001F	Manufacturer specific Recommended for “temporary fault”	Temporary fault
0x0020 – 0x00FF	Reserved for common profiles	Central administrative ^a number to unambiguous distinguish between common profiles
0x0100 – 0x7FFF	Manufacturer specific	Manufacturer specific
0x8000	Data transmission impossible	Data Transmission Impossible
0x8001	Remote mismatch	Remote Mismatch
0x8002	Media redundancy mismatch	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 link.
0x8008	Network component function mismatch	Network functionality problems occur
0x8009	Time mismatch	Time master not existent or precision problems
0x800A – 0x8FFF	Reserved	Unknown error
0x9000 – 0x9FFF	Reserved for profiles	Profile specific
0xA000 – 0xFFFF	Reserved	Unknown error
^a The values of this range are defined as administrative numbers.		

6.2.5.3 Coding of the field ChannelNumber

This field shall be coded as data type Unsigned16 with the values according to Table 376.

Table 376 – ChannelNumber

Value (hexadecimal)	Meaning
0x0000 – 0x7FFF	Manufacturer specific
0x8000	Submodule
0x8001 – 0xFFFF	Reserved

NOTE The ChannelNumber 0x8000 references the submodule itself.

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 effected group

ChannelProperties.Accumulative=0 then the field ChannelNumber shall contain the number of the effected channel

6.2.5.4 Coding of the field ChannelProperties

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: ChannelProperties.Type

This field shall be set according to the Table 377.

Table 377 – ChannelProperties.Type

Value (hexadecimal)	Meaning
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
0x02	2 Bit
0x03	4 Bit
0x04	8 Bit
0x05	16 Bit
0x06	32 Bit
0x07	64 Bit
0x08 – 0xFF	Reserved

Bit 8: ChannelProperties.Accumulative

This field shall be set to one if it is an accumulative diagnosis from more then one channel. Otherwise it shall be set to zero.

Bit 9 – 10: ChannelProperties.Maintenance

This field shall be set according to Table 378 and Table 379.

The dependencies with elements of the field ChannelProperties are specified in Table 378 and the dependencies with Alarmnotification and RecordDataRead(DiagnosisData) in Table 379.

Table 378 – Valid combinations within ChannelProperties

Maintenance-Required Bit 9:	Maintenance-Demanded Bit 10:	Specifier	Meaning	Valid with
0x00	0x00	0x00	All subsequent ^a Diagnosis, MaintenanceRequired, MaintenanceDemanded, and QualifiedDiagnosis disappear	ChannelDiagnosis
		0x01	Diagnosis appears	ChannelDiagnosis, ManufacturerSpecificDiagnosis, ExtChannelDiagnosis, and QualifiedChannelDiagnosis
		0x02	Diagnosis disappears	
		0x03	Diagnosis disappears but other remain	
0x01	0x00	0x00	Reserved	ChannelDiagnosis, ManufacturerSpecificDiagnosis, ExtChannelDiagnosis, and QualifiedChannelDiagnosis
		0x01	MaintenanceRequired appears	
		0x02	MaintenanceRequired disappears	
		0x03	MaintenanceRequired disappears but other remain	
0x00	0x01	0x00	Reserved	ChannelDiagnosis, ManufacturerSpecificDiagnosis, ExtChannelDiagnosis, and QualifiedChannelDiagnosis
		0x01	MaintenanceDemanded appears	
		0x02	MaintenanceDemanded disappears	
		0x03	MaintenanceDemanded disappears but other remain	
0x01	0x01	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 379 – Valid combinations for Alarmnotification and Record-DataRead(DiagnosisData)

Maintenance-Required Bit 9:	Maintenance-Demanded Bit 10:	Specifier	Meaning	Valid with
0x00	0x00	0x00	All subsequent ^a Diagnosis, MaintenanceRequired, MaintenanceDemanded, and QualifiedDiagnosis disappear	Alarmnotification
		0x01	Diagnosis appears	Alarmnotification and Record-DataRead(DiagnosisData)
		0x02	Diagnosis disappears	Alarmnotification RecordDataRead(DiagnosisData) shall only be used in conjunction with ManufacturerSpecificDiagnosis
		0x03	Diagnosis disappears but other remain	Alarmnotification
0x01	0x00	0x00	Reserved	—
		0x01	MaintenanceRequired appears	Alarmnotification and Record-DataRead(DiagnosisData)
		0x02	MaintenanceRequired disappears	Alarmnotification
		0x03	MaintenanceRequired disappears but other remain	
0x00	0x01	0x00	Reserved	—
		0x01	MaintenanceDemanded appears	Alarmnotification and Record-DataRead(DiagnosisData)
		0x02	MaintenanceDemanded disappears	Alarmnotification
		0x03	MaintenanceDemanded disappears but other remain	
0x01	0x01	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 380. The field ChannelProperties.Specifier is related to the addressed channel, ChannelErrorType, and ExtChannelErrorType. The dependencies with elements of the field ChannelProperties are specified in Table 378.

Table 380 – ChannelProperties.Specifier

Value (hexadecimal)	Meaning	Usage
0x00	All subsequent disappears	See Table 378
0x01	Appears	An event appears and/or exists further
0x02	Disappears	An event disappears and/or exists no longer
0x03	Disappears but other remain	An event disappears but there are other

Bit 13 – 15: ChannelProperties.Direction

This field shall be set according to Table 381.

Table 381 – ChannelProperties.Direction

Value (hexadecimal)	Meaning
0x00	Manufacturer specific
0x01	Input
0x02	Output
0x03	Input/Output
0x04 – 0x07	Reserved

6.2.5.5 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 382.

Table 382 – 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 383, Table 384, Table 385, Table 386, Table 387, Table 388, Table 389, Table 390, and Table 391.

Table 383 – ExtChannelErrorType for ChannelErrorType 0 – 0x7FFF

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
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 384 – ExtChannelErrorType for ChannelErrorType “Data transmission impossible”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
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 385 – ExtChannelErrorType for ChannelErrorType “Remote mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Peer Chassis ID mismatch	Alarm/diagnosis
0x8001	Peer Port ID 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	Peer MRRT mismatch	Alarm/diagnosis
0x8007	Peer Line Delay mismatch	Alarm/diagnosis
0x8008	Peer PTCP mismatch ^b	Alarm/diagnosis
0x8009 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

^a This events occurs, if the remote RTClass3_PortStatus.State is not RTCLASS3_RUN. This could also happens, if the remote IRDATAUUID is not equal to the local IRDATAUUID.

^b This events 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 386 – ExtChannelErrorType for ChannelErrorType “Media redundancy mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
0x0001 – 0x7FFF	Manufacturer specific	Alarm/diagnosis
0x8000	Manager role fail	Alarm/diagnosis
0x8001	MRP ring open	Alarm/diagnosis
0x8002	MRRT ring open	Alarm/diagnosis
0x8003	Multiple manager	Alarm/diagnosis
0x8004 – 0x8FFF	Reserved	—
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	—

Table 387 – ExtChannelErrorType for ChannelErrorType “Sync mismatch” and for ChannelErrorType “Time mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
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 388 – ExtChannelErrorType for ChannelErrorType “Isochronous mode mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
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 389 – ExtChannelErrorType for ChannelErrorType “Multicast CR mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
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 390 – ExtChannelErrorType for ChannelErrorType “Fiber optic mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
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 391 – ExtChannelErrorType for ChannelErrorType “Network component function mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
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	—

6.2.5.6 Coding of the field ExtChannelAddValue

This field shall be coded as data type Unsigned32. The value 0 shall be used for no information.

6.2.5.6.1 Coding of the field ExtChannelAddValue for ChannelErrorType = 0 – 0x7FFF

This field shall be coded with the values according to Table 392 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0-0x7FFF.

Table 392 – Values for Accumulative Info

Bitposition	Value (hexadecimal)	Meaning
Bit 0	0x00	ChannelNumber is not effected
	0x01	ChannelNumber is effected
Bit 1	0x00	ChannelNumber + 1 is not effected
	0x01	ChannelNumber + 1 is effected
Bit 2	0x00	ChannelNumber + 2 is not effected
	0x01	ChannelNumber + 2 is effected
...	...	...
Bit30	0x00	ChannelNumber + 30 is not effected
	0x01	ChannelNumber + 30 is effected
Bit31	0x00	ChannelNumber + 31 is not effected
	0x01	ChannelNumber + 31 is effected

6.2.5.6.2 Coding of the field ExtChannelAddValue for ChannelErrorType “Fiber optic mismatch”

This field shall be coded with the values according to Table 393 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0x8007.

Table 393 – Values for “Fiber optic mismatch” – “Power Budget”

Value (hexadecimal)	Meaning
0x00 – 0x3E7	PowerBudget in 0,1dB steps [0..99,9dB]
0x03E8 – 0xFFFFFFFF	Reserved

6.2.5.6.3 Coding of the field ExtChannelAddValue for ChannelErrorType “Network component function mismatch”

This field shall be coded with the values according to Table 394 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0x8008.

Table 394 – 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

6.2.5.6.4 Coding of the field ExtChannelAddValue for ChannelErrorType “Remote mismatch”

This field shall be coded with the values according to Table 395 if the field ExtChannelErrorType contains the value 0x8007 and the field ChannelErrorType contains the value 0x8001.

Table 395 – 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	—

6.2.5.7 Coding of the field QualifiedChannelQualifier

This field shall be coded as data type Unsigned32. The values shall be set according to Table 396. There shall only one bit be set in one QualifiedChannelDiagnosis.

Table 396 – Values for QualifiedChannelQualifier

Bitposition	Value (hexadecimal)	Meaning	Usage
Bit 0	Reserved	—	—
Bit 1	Reserved	—	—
Bit 2	0x00	Qualifier_2 not set	Profile specific
	0x01	Qualifier_2 is set	
Bit 3	0x00	Qualifier_3 not set	
	0x01	Qualifier_3 is set	
...		...	
Bit 30	0x00	Qualifier_30 not set	
	0x01	Qualifier_30 is set	
Bit 31	0x00	Qualifier_31 not set	
	0x01	Qualifier_31 is set	

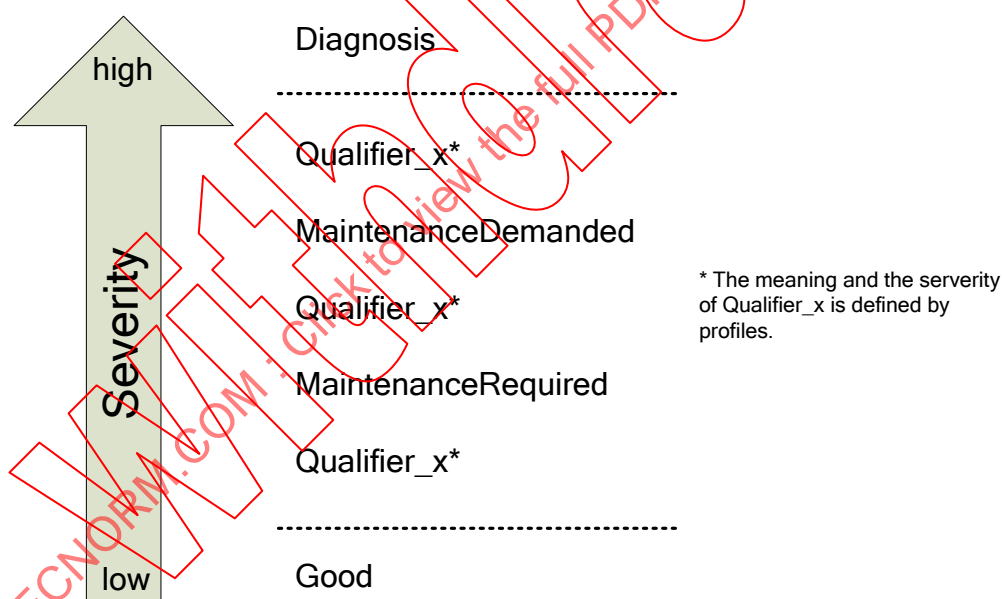
6.2.5.8 Coding of the field MaintenanceStatus

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 397. Otherwise this block shall be omitted.

Table 397 – Values for MaintenanceStatus

Bitposition	Bit name	Value (hexadecimal)	Meaning
Bit 0	MaintenanceRequired	0x00	No maintenance required information available
		0x01	Maintenance required information available
Bit 1	MaintenanceDemanded	0x00	No maintenance demanded information available
		0x01	Maintenance demanded information available
Bit 2	Qualifier_2	0x00	No information available
		0x01	Information available
Bit 3	Qualifier_3	0x00	No information available
		0x01	Information available
...	—	—	...
Bit 30	Qualifier_30	0x00	No information available
		0x01	Information available
Bit 31	Qualifier_31	0x00	No information available
		0x01	Information available

The classification of diagnosis, maintenance and qualified is shown in Figure 55.

**Figure 55 – Classification of diagnosis, maintenance and qualified**

6.2.6 Coding section related to upload and retrieval

6.2.6.1 Coding of the field URRecordIndex

This field shall be coded as data type Unsigned32 with the values according to Table 398.

Table 398 – URRecordIndex

Value (hexadecimal)	Meaning
0x00000000 – 0x0000FFFF	Index of the used record
0x00010000 – 0xFFFFFFFF	Reserved

6.2.6.2 Coding of the field URRecordLength

This field shall be coded as data type Unsigned32 with the values according to Table 399.

Table 399 – 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.

6.2.7 Coding section related to iParameter

6.2.7.1 Coding of the field iPar_Req_Header

This field shall be coded as data type Unsigned32 with the values according to IEC 61784-3-3.

6.2.7.2 Coding of the field Max_Segm_Size

This field shall be coded as data type Unsigned32 with the values according to IEC 61784-3-3.

6.2.7.3 Coding of the field Transfer_Index

This field shall be coded as data type Unsigned32 with the values according to IEC 61784-3-3.

6.2.7.4 Coding of the field Total_iPar_Size

This field shall be coded as data type Unsigned32 with the values according to IEC 61784-3-3.

6.2.8 Coding section related to Physical Device Port Data

6.2.8.1 Coding of the field OwnPortID

This field shall be coded as data type OctetString[8] or OctetString[14] according to 4.9.2.3.

6.2.8.2 Coding of the field LengthOwnPortID

This field shall be coded as data type Unsigned8 and the value 8 or 14 according to 4.9.2.3.

6.2.8.3 Coding of the field NumberOfPeers

This field shall be coded as data type Unsigned8.

6.2.8.4 Coding of the field LengthPeerPortID

This field shall be coded as data type Unsigned8.

6.2.8.5 Coding of the field PeerPortID

This field shall be coded as data type OctetString[255].

6.2.8.6 Coding of the field LengthPeerChassisID

This field shall be coded as data type Unsigned8.

6.2.8.7 Coding of the field PeerChassisID

This field shall be coded as data type OctetString[255].

6.2.8.8 Coding of the field LengthOwnChassisID

This field shall be coded as data type Unsigned8.

6.2.8.9 Coding of the field OwnChassisID

This field shall be coded as data type OctetString[255].

6.2.8.10 Coding of the field LineDelay

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 30: LineDelay.Value

This field shall be set according to the Table 400, Table 401, Figure 18, and Equation (27).

Table 400 – LineDelay.Value with LineDelay.FormatIndicator == 0

Value (hexadecimal)	Meaning
0x00000000	Line delay and cable delay unknown
0x00000001 – 0x7FFFFFFF	Line delay in nanoseconds

Table 401 – 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 the Table 402.

Table 402 – 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
0x01	LineDelay.Value is coded as cable delay	Default, if cable delay is known Shall be used in conjunction with PDPortDataReal

6.2.8.11 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 IEEE 802 MAC address.

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

6.2.9 Coding section related to Physical Device IR Data

6.2.9.1 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 403.

Table 403 – RxPort

Value (hexadecimal)	Meaning
0x00	Local interface
0x01	Port 1
0x02	Port 2
...	...
0xFF	Port 255

6.2.9.2 Coding of the field NumberOfTxPortGroups

This field shall be coded as data type Unsigned8 and shall be set according to Table 404. This field shall only count the succeeding TxPortGroupArray entries.

Table 404 – NumberOfTxPortGroups

Value (hexadecimal)	Meaning
0x01, 0x03, 0x05, 0x07, 0x09, 0x0A, 0x0C, 0x0F	Allowed values
0x11, 0x13, 0x15, 0x17, 0x19, 0x1A, 0x1C, 0x1F	Allowed values
0x21	Allowed values
other	Reserved

6.2.9.3 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

$$N = M_{\text{highest}} \text{ DIV } 8 + 1 \quad (55)$$

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).

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.

NOTE The term DIV stands for division without remainder. The term MOD stands for the remainder of the division.

The TxPortGroup of the devices TxPorts shall be structured in ascending order without gaps. The coding of this field shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0: TxPortEntry_0

This bit shall be set with the values according to Table 405 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 405 – 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 405 if the TxPort with the number that meets $m \text{ MOD } 8 = 1$ is present. A padding bit shall be used if no TxPort is present.

Bit 2: TxPortEntry_2

This bit shall be set with the values according to Table 405 if the TxPort with the number that meets $m \text{ MOD } 8 = 2$ is present. A padding bit shall be used if no TxPort is present.

Bit 3: TxPortEntry_3

This bit shall be set with the values according to Table 405 if the TxPort with the number that meets $m \text{ MOD } 8 = 3$ is present. A padding bit shall be used if no TxPort is present.

Bit 4: TxPortEntry_4

This bit shall be set with the values according to Table 405 if the TxPort with the number that meets $m \text{ MOD } 8 = 4$ is present. A padding bit shall be used if no TxPort is present.

Bit 5: TxPortEntry_5

This bit shall be set with the values according to Table 405 if the TxPort with the number that meets $m \text{ MOD } 8 = 5$ is present. A padding bit shall be used if no TxPort is present.

Bit 6: TxPortEntry_6

This bit shall be set with the values according to Table 405 if the TxPort with the number that meets $m \text{ MOD } 8 = 6$ is present. A padding bit shall be used if no TxPort is present.

Bit 7: TxPortEntry_7

This bit shall be set with the values according to Table 405 if the TxPort with the number that meets $m \text{ MOD } 8 = 7$ is present. A padding bit shall be used if no TxPort is present.

Where m is the number of the current TxPort with $1 \leq m \leq N$.

6.2.9.4 Coding of the field FrameDetails

This field shall be coded according to 3.7.3.3 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 406.

Table 406 – FrameDetails.SyncFrame

Value (hexadecimal)	Meaning	Description
0x00	No sync frame	—
0x01	Primary sync frame	Primary and secondary sync frame use the same FrameID. At one time only one of this frames shall be valid.
0x02	Secondary sync frame	
0x03	Reserved	—

Bit 2 – 3: FrameDetails.MeaningFrameSendOffset

This field shall be coded with the values according to Table 407.

Table 407 – FrameDetails.MeaningFrameSendOffset

Value (hexadecimal)	Meaning
0x00	Field FrameSendOffset specifies the point of time for receiving or transmitting a frame
0x01	Field FrameSendOffset specifies the beginning of the RT_CLASS_3 interval within a phase
0x02	Field FrameSendOffset specifies the ending of the RT_CLASS_3 interval within a phase
0x03	Reserved

Bit 4 – 7: FrameDetails.reserved

This field shall be set to zero.

6.2.9.5 Coding of the field AdjustProperties

This field shall be coded as data type Unsigned16 with the value zero.

6.2.9.6 Coding of the field MAUType

This field shall be coded as data type Unsigned16 with the values according to IETF RFC 4836, Table 408 and Table 409.

Table 408 – MAUType

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	—
0x0005	10BASET	PDPortDataReal
0x000A	10BASETXHD	PDPortDataReal
0x000B	10BASETXFD	PDPortDataReal
0x000C	10BASEFLHD	PDPortDataReal
0x000D	10BASEFLFD	PDPortDataReal
0x000F	100BASETXHD	PDPortDataReal
0x0010	Default 100BASETXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0011	100BASEFXHD	PDPortDataReal
0x0012	100BASEFXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0015	1000BASEXHD	PDPortDataReal
0x0016	1000BASEXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0017	1000BASELXHD	PDPortDataReal
0x0018	1000BASELXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0019	1000BASESXHD	PDPortDataReal
0x001A	1000BASESXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x001D	1000BASETHD	PDPortDataReal
0x001E	1000BASETFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x001F	10GigBASEFX	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x002E	100BASELX10	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0036	100BASEPXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
Useable well known numbers	Speed of 100 Mbit/s (and more) and duplexity full	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
Unuseable well known numbers	Speed less than 100 Mbit/s or duplexity half	PDPortDataReal
other	Reserved	—

Table 409 – Valid combinations between MAUType and LinkState

LinkState.Port	LinkState.Link	MAUType	Usage
Unknown Disabled / Discarding Blocking Listening Learning Forwarding Broken	Up (ready to pass packets)	10BASET	PDPortDataReal
		10BASETXHD	
		10BASETXFD	
		10BASEFLHD	
		10BASEFLFD	
		100BASETXHD	
		100BASETXFD (Default)	
		100BASEFXHD	
		100BASEFXFD	
		1000BASEXHD	
		1000BASEXFD	
		1000BASELXHD	
		1000BASELXFD	
		1000BASESXHD	
		1000BASESXFD	
		1000BASETHD	
		1000BASETFD	
		10GigBASEFX	
		100BASELX10	
		100BASEPXFHD	
		Useable well known numbers	
		Unuseable well known numbers	
Reserved	Down	Reserved	
	Testing (in some test mode)		
	Unknown (status can not determined)		
Reserved	Reserved		

6.2.9.7 Coding of the field CheckSyncMode

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0: CheckSyncMode.CableDelay

This field shall be set according to the Table 410.

Table 410 – 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 the Table 411.

Table 411 – 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 3.7.3.2.

6.2.9.8 Coding of the field MAUTypeMode

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0: MAUTypeMode.Check

This field shall be set according to the Table 412.

Table 412 – MAUTypeMode.Check

Value (hexadecimal)	Meaning	Usage
0x00	OFF	No check
0x01	ON	Check MAU type difference between local and remote detected value.

Bit 1 – 15: MAUTypeMode.reserved

This field shall be set according to 3.7.3.2.

6.2.9.9 Coding of the field DomainBoundaryIngress

The coding of this field shall be according to 3.7.3.5 and the individual bits shall be coded with the values according to Table 413.

Table 413 – DomainBoundaryIngress

Bit	Value	Meaning
0	1	Block an incoming frame with the multicast MAC address 01-0E-CF-00-04-00, 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-00-04-00, 01-0E-CF-00-04-20, 01-0E-CF-00-04-40, and 01-0E-CF-00-04-80
...	1	Block an incoming frame with the multicast MAC address 01-0E-CF-00-04-xx, 01-0E-CF-00-04-xx, 01-0E-CF-00-04-xx, and 01-0E-CF-00-04-xx
	0	Do not block an incoming frame with the multicast MAC address 01-0E-CF-00-04-xx, 01-0E-CF-00-04-xx, 01-0E-CF-00-04-xx, and 01-0E-CF-00-04-xx
31	1	Block an incoming frame with the multicast MAC address 01-0E-CF-00-04-1F, 01-0E-CF-00-04-3F, 01-0E-CF-00-04-5F, and 01-0E-CF-00-04-9F
	0	Do not block an incoming frame with the multicast MAC address 01-0E-CF-00-04-1F, 01-0E-CF-00-04-3F, 01-0E-CF-00-04-5F, and 01-0E-CF-00-04-9F

6.2.9.10 Coding of the field DomainBoundaryEgress

The coding of this field shall be according to 3.7.3.5 and the individual bits shall be coded with the values according to Table 414.

Table 414 – DomainBoundaryEgress

Bit	Value	Meaning
0	1	Block an outgoing frame with the multicast MAC address 01-0E-CF-00-04-00, 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-00-04-00, 01-0E-CF-00-04-20, 01-0E-CF-00-04-40, and 01-0E-CF-00-04-80
...	1	Block an outgoing frame with the multicast MAC address 01-0E-CF-00-04-xx, 01-0E-CF-00-04-xx, 01-0E-CF-00-04-xx, and 01-0E-CF-00-04-xx
	0	Do not block an outgoing frame with the multicast MAC address 01-0E-CF-00-04-xx, 01-0E-CF-00-04-xx, 01-0E-CF-00-04-xx, and 01-0E-CF-00-04-xx
31	1	Block an outgoing frame with the multicast MAC address 01-0E-CF-00-04-1F, 01-0E-CF-00-04-3F, 01-0E-CF-00-04-5F, and 01-0E-CF-00-04-9F
	0	Do not block an outgoing frame with the multicast MAC address 01-0E-CF-00-04-1F, 01-0E-CF-00-04-3F, 01-0E-CF-00-04-5F, and 01-0E-CF-00-04-9F

6.2.9.11 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 415.

Table 415 – 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

This shall be applied for the first 32 RT_CLASS_2 multicast addresses from 01-0E-CF-00-02-00 to 01-0E-CF-00-02-1F.

6.2.9.12 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 415.

Table 416 – 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 frames (egress filter).
	0	The PTCP ASE shall send PTCP_DELAY frames for this port.
...	0	Reserved
31	0	Reserved

6.2.9.13 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 415.

Table 417 – 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

6.2.9.14 Coding of the field LinkState

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: LinkState.Link

This field shall be set according to Table 418.

Table 418 – 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 can not 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 419.

Table 419 – 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	—

6.2.9.15 Coding of the field MediaType

This field shall be coded as data type Unsigned32 with the values according to Table 420.

Table 420 – MediaType

Value (hexadecimal)	Meaning	Usage
0x00	Unknown	PDPortDataReal
0x01	Copper cable	PDPortDataReal
0x02	Fiber optic cable	PDPortDataReal
0x03	Radio communication	PDPortDataReal
0x04 – 0xFFFFFFFF	Reserved	—

6.2.9.16 Coding of the field MaxBridgeDelay

This field shall be coded as data type Unsigned32 according to Table 421. Figure 19 shows the meaning of MaxBridgeDelay.

Table 421 – MaxBridgeDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x3B9AC9FF	From engineering used bridge delay for RT_CLASS_3 calculation
0x3B9ACA00 – 0xFFFFFFFF	Reserved

6.2.9.17 Coding of the field NumberOfPorts

This field shall be coded as data type Unsigned32. This field shall be coded with the values according to Table 422.

Table 422 – NumberOfPorts

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x000000FF	Number of following port entries
0x00000100 – 0xFFFFFFFF	Reserved

6.2.9.18 Coding of the field MaxPortTxDelay

This field shall be coded as data type Unsigned32 according to Table 423. Figure 19 shows the meaning of MaxPortTxDelay.

Table 423 – MaxPortTxDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x3B9AC9FF	From engineering used port transmit delay for RT_CLASS_3 calculation
0x3B9ACA00 – 0xFFFFFFFF	Reserved

6.2.9.19 Coding of the field MaxPortRxDelay

This field shall be coded as data type Unsigned32 according to Table 424. Figure 19 shows the meaning of MaxPortRxDelay.

Table 424 – MaxPortRxDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x3B9AC9FF	From engineering used port receive delay for RT_CLASS_3 calculation
0x3B9ACA00 – 0xFFFFFFFF	Reserved

6.2.9.20 Coding of the field MaxLineRxDelay

This field shall be coded as data type Unsigned32 according to Table 425. Figure 19 shows the meaning of MaxLineRxDelay.

Table 425 – 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

6.2.9.21 Coding of the field YellowTime

This field shall be coded as data type Unsigned32 according to Table 426 in octets. Figure 56 shows the meaning of YellowTime.

The YELLOW period allows a bridge to optimized the forwarding of frames. If in GREEN and outside YELLOW a bridge should use “cut through”.

Table 426 – YellowTime

Value (hexadecimal)	Meaning	Usage
0x00 – 0x5FF	Reserved	—
0x0600	Default 1 536 octets frame size	100 Mbit/s: 122 880 ns 1 Gbit/s: 12 288 ns
0x0601 – 0xFFFFFFFF	Reserved	—

NOTE An implementation may add a safety margin to cover the local issues.

6.2.9.22 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 427 and Table 429.

Table 427 – StartOfRedFrameID

Value (hexadecimal)	Meaning
0x0000 – 0x007F	Reserved
0x0080 – 0x7FFF	Mandatory
0x8000 – 0xFFFF	Reserved

6.2.9.23 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 428 and Table 429.

Table 428 – EndOfRedFrameID

Value (hexadecimal)	Meaning
0x0000 – 0x007F	Reserved
0x0080 – 0x7FFF	Mandatory
0x8000 – 0xFFFF	Reserved

Table 429 – Dependencies of StartOfRedFrameID and EndOfRedFrameID

Value (hexadecimal)	Meaning
StartOfRedFrameID < EndOfRedFrameID	Valid
StartOfRedFrameID ≥ EndOfRedFrameID	Not used

6.2.9.24 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 430.

Table 430 – NumberOfAssignments

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x00000004	Mandatory, if only phases with reserved periods (RED or ORANGE) exists, four entries shall be supported. Number of following assignment entries
0x00000005	Mandatory, if phases without reserved periods (RED or ORANGE) exists, five entries shall be supported. Number of following assignment entries
0x00000006 – 0x00000010	Optional Number of following assignment entries
0x00000011 – 0xFFFFFFFF	Reserved

6.2.9.25 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 431.

Table 431 – 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

6.2.9.26 Coding of the field PhaseAssignment

The field PhaseAssignment shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 3: AssignedValueForReservedBegin

This bit shall be set with the values according to Table 432.

Table 432 – 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 433.

Table 433 – 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 434.

Table 434 – 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.

6.2.9.27 Coding of the field RedOrangePeriodBegin

This field shall be coded as data type Unsigned32 and set according to Table 435 and Table 436. The time base is 1 ns.

Table 435 – Values of RedOrangePeriodBegin

Value (hexadecimal)	Meaning	Use
0 – 0x003D08FF	Relative offset to the start of the related cycle	Optional
0x003D0900 – 0xFFFFFFFF	Reserved	—

Table 436 – Dependencies of RedOrangePeriodBegin, OrangePeriodBegin, and GreenPeriodBegin

Value (hexadecimal)	Meaning
RedOrangePeriodBegin < OrangePeriodBegin < GreenPeriodBegin	A RED and ORANGE period is defined
RedOrangePeriodBegin = OrangePeriodBegin < GreenPeriodBegin	Only a ORANGE period is defined
RedOrangePeriodBegin < OrangePeriodBegin = GreenPeriodBegin	Only a RED period is defined
RedOrangePeriodBegin = OrangePeriodBegin = GreenPeriodBegin = 0	No reserved period is defined
RedOrangePeriodBegin > OrangePeriodBegin	Not used
RedOrangePeriodBegin > GreenPeriodBegin	Not used
OrangePeriodBegin > GreenPeriodBegin	Not used

6.2.9.28 Coding of the field OrangePeriodBegin

This field shall be coded as data type Unsigned32 and set according to Table 436 and Table 437. The time base is 1 ns.

Table 437 – Values of OrangePeriodBegin

Value (hexadecimal)	Meaning	Use
0 – 0x003D08FF	Relative offset to the start of the related cycle	Optional
0x003D0900 – 0xFFFFFFFF	Reserved	—

6.2.9.29 Coding of the field GreenPeriodBegin

This field shall be coded as data type Unsigned32 and set according to Table 436 and Table 438. The time base is 1 ns.

Table 438 – Values of GreenPeriodBegin

Value (hexadecimal)	Meaning	Use
0 – 0x003D08FF	Relative offset to the start of the related cycle	Optional
0x003D0900 – 0xFFFFFFFF	Reserved	—

6.2.9.30 Coding of the field EtherType

This field shall be coded as Unsigned16 as described for the field LT in 4.2.2.40. This specification uses the value according to Table 439.

Table 439 – EtherType

Value (hexadecimal)	Meaning
0x8892	RT_CLASS_x

6.2.10 Coding section related to Physical Sync Data**6.2.10.1 Coding of the field PTCPSubdomainID**

This field shall be coded as data type UUID. The value NULL indicates no synchronisation within the Read Real Sync Data service.

6.2.10.2 Coding of the field PTCPLengthSubdomainName

This field shall be coded as data type Unsigned8.

6.2.10.3 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.

6.2.10.4 Coding of the field SyncProperties

This field shall be coded according to 3.7.3.4 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 440.

Table 440 – SyncProperties.Role

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	—
0x01	External sync	Clock or Time Slave
0x02	Internal sync	Clock or Time 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 441.

Table 441 – SyncProperties.SyncID

Value (hexadecimal)	Meaning
0x00	SyncID 0
0x01	SyncID 1
...	...
0x1F	SyncID 31

Bit 13 – 15: SyncProperties.reserved

This field shall be set to zero.

6.2.10.5 Coding of the field ReservedIntervalBegin

This field shall be coded as data type Unsigned32. The time base is one ns. The definition of this field shall be according to Figure 56 and Table 442. This field shall only be evaluated if SyncProperties.SyncID is set to zero.

6.2.10.6 Coding of the field ReservedIntervalEnd

This field shall be coded as data type Unsigned32. The time base is one ns. The definition of this field shall be according to Figure 56 and Table 442. This field shall only be evaluated if SyncProperties.SyncID is set to zero.

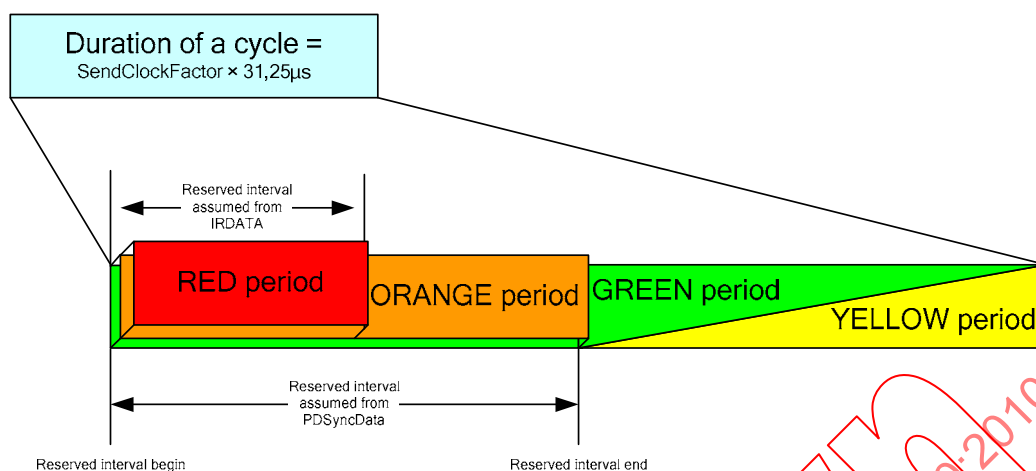


Figure 56 – Definition of the reserved interval

Table 442 – Dependencies of ReservedIntervalBegin and ReservedIntervalEnd

Value (hexadecimal)	Meaning
ReservedIntervalBegin < ReservedIntervalEnd	A ORANGE period is defined
ReservedIntervalBegin = ReservedIntervalEnd = 0	No reserved period is defined
ReservedIntervalBegin > ReservedIntervalEnd	Not used

6.2.10.7 Coding of the field SyncSendFactor

This field shall be coded as data type Unsigned32. The time base is 31,25 µs. The value range shall be according to Table 443.

Table 443 – 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	—	—

NOTE 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 Equation (56).

$$\text{SyncSendInterval} = \text{SyncSendFactor} \times 31,25 \mu\text{s} \quad (56)$$

6.2.10.8 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 444.

Table 444 – 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 Equation (57).

$$\text{Timeout} = \text{PTCPTimeoutFactor} \times \text{SyncSendFactor} \times 31,25 \mu\text{s} \quad (57)$$

6.2.10.9 Coding of the field PTCPTakeoverTimeoutFactor

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 445.

Table 445 – PTCPTakeoverTimeoutFactor

Value (hexadecimal)	Meaning
0x0000	Disabled
0x0001 – 0x0002	Optional
0x0003	Default, mandatory
0x0004 – 0x000F	Mandatory
0x0010 – 0x01FF	Optional
0x0200 – 0xFFFF	Reserved

Each SyncProperties.SyncID shall require its own PTCPTakeoverTimeoutFactor. The Timeout shall be calculated according to Equation (58).

$$\text{Timeout} = \text{PTCPTakeoverTimeoutFactor} \times \text{SyncSendFactor} \times 31,25 \mu\text{s} \quad (58)$$

6.2.10.10 Coding of the field PTCPMasterStartupTime

This field shall be coded as data type Unsigned16. The time base is one second. The value range shall be according to Table 446.

Table 446 – PTCPMasterStartupTime

Value (hexadecimal)	Meaning
0x0000	Disabled
0x0001 – 0x0004	Optional
0x0005 – 0x0009	Mandatory
0x000A	Default, mandatory
0x000B – 0x003C	Mandatory
0x003D – 0x012C	Optional
0x012D – 0xFFFF	Reserved

Each SyncProperties.SyncID shall require its own PTCPMasterStartupTime.

6.2.10.11 Coding of the field PLLWindow

This field shall be coded as data type Unsigned32. The time base is one nanosecond. The value range shall be according to Table 447.

Table 447 – PLLWindow

Value (hexadecimal)	Meaning	SyncID (Clock)	SyncID (Time)
0x00	Disabled	—	—
0x0001 – 0x03E7	Optional	—	—
0x03E8	Default	Mandatory (1 μ s)	Optional (1 μ s)
0x03E9 – 0x270F	Optional	—	—
0x2710	Default	Optional (10 μ s)	Optional (10 μ s)
0x2710 – 0x000F423F	Optional	—	—
0x000F4240	Default	Optional (1 ms)	Mandatory (1 ms)
0x000F423F – 0x98967F	Optional	—	—
0x989680	Default	Optional (10 ms)	Optional (10 ms)
0x989681 – 0xFFFFFFFF	Reserved	—	—

Each SyncProperties.SyncID shall require its own PLLWindow.

The definition of the PLLWindow is shown in Figure 57.

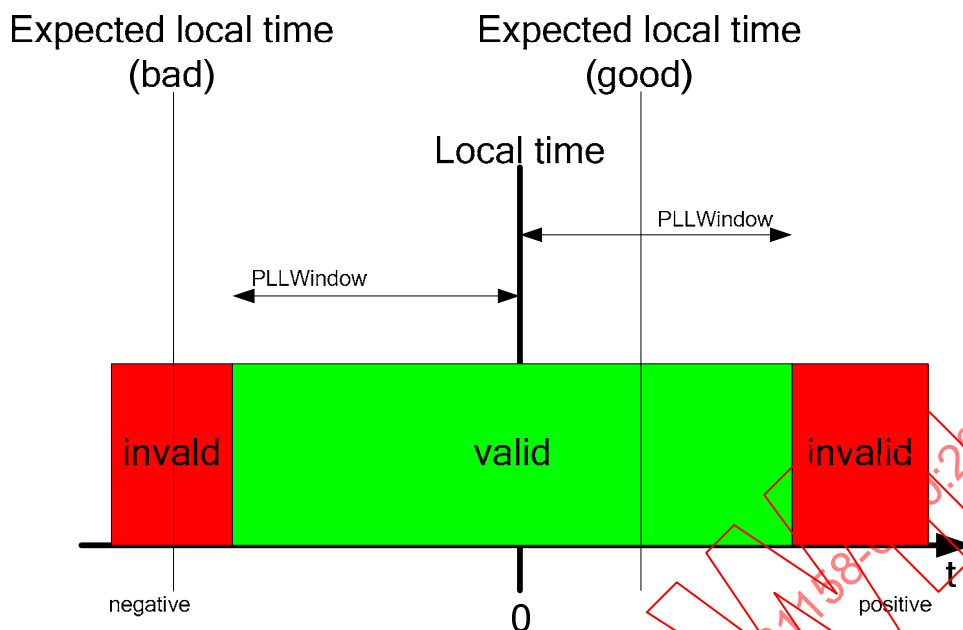


Figure 57 – Definition of PLL window

6.2.11 Coding section related to Isochrone Mode Data

6.2.11.1 Coding of the field TimeDataCycle

This field shall be coded as data type Unsigned16. The time base is 31,25 μ s. The value range shall be according to Table 448.

Table 448 – TimeDataCycle

Value (hexadecimal)	Meaning
0x00	Reserved
0x01 – 0x0400	Optional
0x401 – 0xFFFF	Reserved

6.2.11.2 Coding of the field TimeIOInput

This field shall be coded as data type Unsigned32. The time base is ns. The value range shall be according to Table 449.

Table 449 – TimeIOInput

Value (decimal)	Meaning
1 – 32 000 000	Optional
other	Reserved

6.2.11.3 Coding of the field TimeIOOutput

This field shall be coded as data type Unsigned32. The time base is ns. The value range shall be according to Table 450.

Table 450 – TimeIOOutput

Value (decimal)	Meaning
1 – 32 000 000	Optional
other	Reserved

6.2.11.4 Coding of the field TimeIOInputValid

This field shall be coded as data type Unsigned32. The time base is ns. The value range shall be according to Table 451.

Table 451 – TimeIOInputValid

Value (decimal)	Meaning
1 – 32 000 000	Optional
other	Reserved

6.2.11.5 Coding of the field TimeIOOutputValid

This field shall be coded as data type Unsigned32. The time base is ns. The value range shall be according to Table 452.

Table 452 – TimeIOOutputValid

Value (decimal)	Meaning
1 – 32 000 000	Optional
other	Reserved

6.2.11.6 Coding of the field ControllerApplicationCycleFactor

This field shall be coded as data type Unsigned16. The value range shall be according to Table 453.

Table 453 – ControllerApplicationCycleFactor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x0400	Optional
0x0401 – 0xFFFF	Reserved

6.2.12 Coding section related to Media Redundancy**6.2.12.1 Coding of the field MRP_Role**

This field shall be coded as data type Unsigned16 and shall be set according to Table 454.

Table 454 – MRP_Role

Value (hexadecimal)	Meaning
0x0000	Media Redundancy disabled
0x0001	Media Redundancy Client
0x0002	Media Redundancy Manager
0x0003 – 0xFFFF	Reserved

6.2.12.2 Coding of the field MRP_RTMode

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0: MRP_RTMode.RTClass1_2

This field shall be set according to the Table 455.

Table 455 – MRP_RTMode.RTClass1_2

Value (hexadecimal)	Meaning
0x00	OFF RT_CLASS_1 and RT_CLASS_2 redundancy mode deactivated MRRT deactivated
0x01	ON RT_CLASS 1 and RT_CLASS 2 redundancy mode activated MRRT activated

Bit 1 – 23: MRP_RTMode.reserved_1

This field shall be set according to 3.7.3.2.

Bit 24 – 31: MRP_RTMode.reserved_2

This field shall be set to zero.

6.2.12.3 Coding of the field MRP_TOPchgT

This field defines the common point of time that shall be used to invoke the Flush Filtering Data Base service and shall be coded as data type Unsigned16. The value shall be set according to Table 456 with a time base of 10 ms.

Table 456 – MRP_TOPchgT

Value (decimal)	Meaning	Usage
0	0 ms	Clear FDB immediately
1	10 ms	Mandatory
2 – 100	20 ms – 1 s	Optional
101 – 65 535	Reserved	—

6.2.12.4 Coding of the field MRP_TOPNRmax

This field shall be coded as data type Unsigned16 and set according to Table 457.

Table 457 – 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	—

6.2.12.5 Coding of the field MRP_TSTshortT

This field shall be coded as data type Unsigned16 and set according to Table 458.

Table 458 – MRP_TSTshortT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 9	1 – 9 ms	Optional (short test interval)
10	10 ms	Mandatory (200 ms reconfiguration time)
10 – 500	10 – 500 ms	Optional (short test interval)
501 – 65 535	Reserved	—

6.2.12.6 Coding of the field MRP_TSTdefaultT

This field shall be coded as data type Unsigned16 and set according to Table 459.

Table 459 – MRP_TSTdefaultT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 19	1 – 19 ms	Optional (default test interval)
20	20 ms	Mandatory (200 ms reconfiguration time)
21 – 1 000	21 ms – 1 s	Optional (default test interval)
1 001 – 65 535	Reserved	—

6.2.12.7 Coding of the field MRP_TSTNRmax

This field shall be coded as data type Unsigned16 and set according to Table 460.

Table 460 – 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	—

6.2.12.8 Coding of the field MRP_LNKdownT

This field shall be coded as data type Unsigned16. The coding shall be according to Table 461.

Table 461 – MRP_LNKdownT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 19	1 – 19 ms Link Down interval	Optional
20	20 ms Link Down interval	Mandatory
21 – 1 000	21 – 1 000 ms Link Down interval	Optional
1 001 – 65 535	Reserved	—

6.2.12.9 Coding of the field MRP_LNKupT

This field shall be coded as data type Unsigned16 according to Table 462.

Table 462 – MRP_LNKupT

Value (decimal)	Meaning	Usage
0	Reserved	—
1 – 19	1 – 19 ms Link Up interval	Optional
20	20 ms Link Up interval	Mandatory
21 – 1 000	21 – 1 000 ms Link Up interval	Optional
1 001 – 65 535	Reserved	—

6.2.12.10 Coding of the field MRP_LNKNRmax

This field shall be coded as data type Unsigned16 according to Table 463.

Table 463 – 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	

6.2.12.11 Coding of the field MRP_RTState

This field shall be coded as data type Unsigned16 and set according to Table 464.

Table 464 – MRP_RTState

Value (hexadecimal)	Meaning
0x0000	RT media redundancy lost
0x0001	RT media redundancy available
0x0002 – 0xFFFF	Reserved

6.2.12.12 Coding of the field MRP_Check

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0: MRP_Check.MediaRedundancyManager

This field shall be set according to the Table 465.

Table 465 – MRP_Check.MediaRedundancyManager

Value (hexadecimal)	Meaning
0x00	OFF
0x01	ON

Bit 1: MRP_Check.MRP_DomainUUID

This field shall be set according to the Table 466.

Table 466 – MRP_Check.MRP_DomainUUID

Value (hexadecimal)	Meaning
0x00	OFF
0x01	ON Check MRP_DomainUUID vs. LLDP_PNIO_MRPPORTSTATUS

Bit 2 – 23: MRP_Check.reserved_1

This field shall be set according to 3.7.3.2.

Bit 24 – 31: MRP_Check.reserved_2

This field shall be set to zero.

6.2.13 Coding section related to fiber optics

6.2.13.1 Coding of the field VendorBlockType

This field shall be coded as data type Unsigned16 and set according to Table 467.

Table 467 – VendorBlockType

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Vendor specific

6.2.13.2 Coding of the field FiberOpticType

This field shall be coded as data type Unsigned32 and set according to Table 468.

Table 468 – 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, NA = 0,5
0x00000005	SI-PCF, NA = 0,36
0x00000006	LowNA-POF, NA = 0,3
0x00000007	GI-POF
0x00000008 – 0x0000007F	Reserved
0x00000080 – 0x000000FF	Vendor specific
0x00000100 – 0xFFFFFFFF	Reserved

6.2.13.3 Coding of the field FiberOpticCableType

This field shall be coded as data type Unsigned32. The coding shall be according to Table 469.

Table 469 – FiberOpticCableType

Value (hexadecimal)	Meaning
0x0000	No cable specified
0x0001	Inside/outside cable, fixed installation
0x0002	Inside/outside cable, flexible installation
0x0003	Outdoor cable, fixed installation
0x0004 – 0xFFFFFFFF	Reserved

6.2.13.4 Coding of the field FiberOpticPowerBudgetType

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 30: FiberOpticPowerBudgetType.Value

This field shall be set according to the Table 470.

Table 470 – FiberOpticPowerBudgetType.Value

Value (hexadecimal)	Meaning	Usage
0	PowerBudget in 0,1 dB steps	(0 dB): mandatory for maintenance demanded
0x0001 – 0x0013	PowerBudget in 0,1 dB steps	Optional
0x0014	PowerBudget in 0,1 dB steps	(2 dB): mandatory for maintenance required
0x0015 – 0x03E7	PowerBudget in 0,1 dB steps	Optional
0x03E8 – 0x7FFFFFFF	Reserved	Reserved

Bit 31: FiberOpticPowerBudgetType.CheckEnable

This field shall be set according to the Table 470.

Table 471 – FiberOpticPowerBudgetType.CheckEnable

Value (hexadecimal)	Meaning
0x0	OFF
0x1	ON Comparison value is FiberOpticPowerBudgetType.Value

6.2.14 Coding section related network components**6.2.14.1 Coding of the field NCDropBudgetType**

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 30: NCDropBudgetType.Value

This field shall be set according to the Table 472.

Table 472 – NCDropBudgetType.Value

Value (hexadecimal)	Meaning	Usage
0	Reserved	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	Reserved

Bit 31: NCDropBudgetType.CheckEnable

This field shall be set according to the Table 473.

Table 473 – 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 58 and Figure 59.

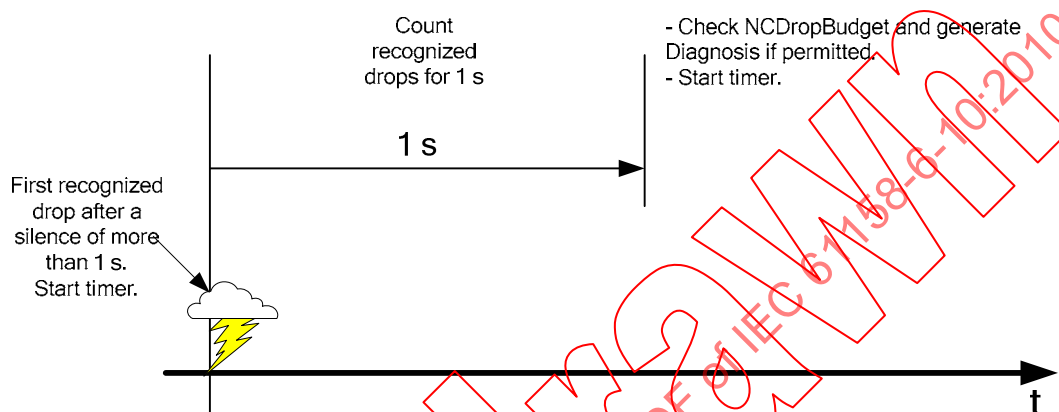


Figure 58 – Detection of dropped frames — appear

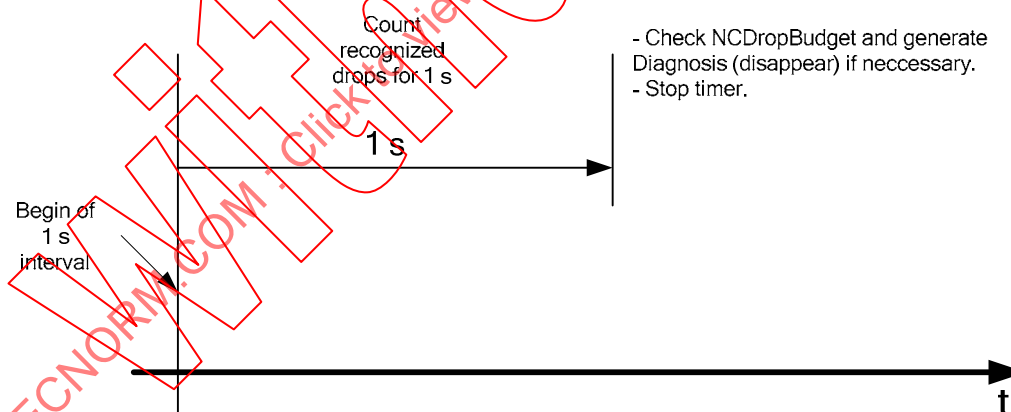


Figure 59 – Detection of dropped frames — disappear

6.2.15 Coding section related to fast start up

6.2.15.1 Coding of the field FSHelloMode

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 1: FSHelloMode.Mode

This field shall be set according to the Table 474.

Table 474 – FSHelloMode.Mode

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	Send DCP_Hello.req on LinkUp	—
0x02	Send DCP_Hello.req on LinkUp after HelloDelay	—
0x03	Reserved	Reserved

Bit 2 – 23: FSHelloMode.reserved_1

This field shall be set according to 3.7.3.2.

Bit 24 – 31: FSHelloMode.reserved_2

This field shall be set to zero.

6.2.15.2 Coding of the field FSHelloInterval

This field shall be coded as data type Unsigned32. The coding shall be according to Table 475.

Table 475 – 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	—

6.2.15.3 Coding of the field FSHelloRetry

This field shall be coded as data type Unsigned32. The coding shall be according to Table 476.

Table 476 – 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	—

6.2.15.4 Coding of the field FSHelloDelay

This field shall be coded as data type Unsigned32. The coding shall be according to Table 477.

Table 477 – 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	—

6.2.15.5 Coding of the field FSParameterMode

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning.

Bit 0 – 1: FSParameterMode.Mode

This field shall be set according to the Table 478.

Table 478 – FSParameterMode.Mode

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	ON	—
0x02	Reserved	—
0x03	Reserved	—

Bit 2 – 23: FSParameterMode.reserved_1

This field shall be set according to 3.7.3.2.

Bit 24 – 31: FSParameterMode.reserved_2

This field shall be set to zero.

6.2.15.6 Coding of the field FSPParameterUUID

This field shall be coded as data type UUID. The coding shall be according to Table 479.

Table 479 – 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.

6.2.16 PDU checking rules

6.2.16.1 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.

6.2.16.2 IODConnectReq

6.2.16.2.1 ArgsLength

This field shall be checked according to Table 480.

Table 480 – ArgsLength check

	Parameter	Checking rules	Behaviour 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)	—

	Parameter	Checking rules	Behaviour on match
NDRDataRequest	ArgsLength	!= Sum of all BlockLength + Number of Blocks * 4	RMPM: ArgsLength invalid
IODConnectReq	ARBlockReq	Not first block	RSP-, 0xDB, 0x81, 0x01, 0
IODConnectReq		Case ARProperties.DeviceAccess == 1: other blocks than ARBlockReq	RMPM: 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	RMPM: IOCR missing
IODConnectReq	AlarmCRBlockReq	Case ARProperties.Device- Access == 0: Number of AlarmCRBlockReq != 1	RMPM: Wrong AlarmCRBlock count

6.2.16.2.2 ARBlockReq

This field shall be checked according to Table 481.

Table 481 – ARBlockReq – request check

Parameter	Checking rules	Behaviour 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 != NULL)	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	== reserved	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.State	If ARType == (IOCARSingle OR IOSAR) AND ARProperties.State != Primary	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.Supervisor- TakeoverAllowed	Don't care	—
ARProperties.Parametrization Server	== External PmServer	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	== 0 to 0x03FF	RSP-, 0xDB, 0x81, 0x01, 11
StationNameLength	== 0 OR > 240	RSP-, 0xDB, 0x81, 0x01, 12
CMinitiatorStationName	NOT (VisibleString, according to RFC 3490)	RSP-, 0xDB, 0x81, 0x01, 13

6.2.16.2.3 IOCRBlockReq

This field shall be checked according to Table 482.

Table 482 – IOCRBlockReq – request check

Parameter	Checking rules	Behaviour 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 correspondig MCRBlock)	RSP-, 0xDB, 0x81, 0x02, 5
LT	Case IOCRProperties.RTClass != 0x04: != 0x8892 Case IOCRProperties.RTClass == 0x04: != 0x0800	RSP-, 0xDB, 0x81, 0x02, 6
IOCRProperties.RTClass	!= 0x01 to 0x04 OR not supported in GSDML	RSP-, 0xDB, 0x81, 0x02, 7
IOCRProperties.reserved_1	!= 0x0	RSP-, 0xDB, 0x81, 0x02, 7
IOCRProperties. MediaRedundancy	!= 0x0 OR 0x01 OR not supported in GSDML	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) AND DataLength != 40 to 1 440) OR (IOCRProperties.RTClass == 0x03 AND DataLength != 0 to 1 440)	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	Behaviour on match
FrameID	If IOCRTYPE==INPUT_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) OR (IOCRProperties.RTClass == 0x04 AND FrameID!=0xC000 to 0xF7FF))	RSP-, 0xDB, 0x81, 0x02, 9
SendClockFactor	!=1 to 128 OR not supported in GSDML OR (IOCRProperties.RTClass == 0x3 AND SendClockFactor != LocalClock) OR (IOCRProperties.RTClass == 0x3 AND SendClockFactor * ReductionRatio < LocalClock)	RSP-, 0xDB, 0x81, 0x02, 10
ReductionRatio	If IOCRProperties.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)	RSP-, 0xDB, 0x81, 0x02, 11
ReductionRatio	If IOCRProperties.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
Sequence	not checked	—
FrameSendOffset	If (FrameSendOffset ≥ SendClockFactor * 31 250 ns) OR ((FrameSendOffset >= 0x003D0900) AND (FrameSendOffset <= 0xFFFFFFFF))	RSP-, 0xDB, 0x81, 0x02, 14

Parameter	Checking rules	Behaviour on match
WatchdogFactor	If WatchdogFactor != 0x0001 to 0x1E00 OR (IOCRProperties.RTClass == 0x04 AND WatchdogFactor * ReductionRatio * SendClockFactor * 31,25 > 61 440 000) OR (IOCRProperties.RTClass == 0x01 to 0x03 AND WatchdogFactor * ReductionRatio * SendClockFactor * 31,25 > 1 920 000)	RSP-, 0xDB, 0x81, 0x02, 15
DataHoldFactor	If DataHoldFactor != 0x0001 to 0x1E00 OR (IOCRProperties.RTClass == 0x04 AND DataHoldFactor * ReductionRatio * SendClockFactor * 31,25 > 61 440 000) OR (IOCRProperties.RTClass == 0x01 to 0x03 AND DataHoldFactor * ReductionRatio * SendClockFactor * 31,25 > 1 920 000)	RSP-, 0xDB, 0x81, 0x02, 16
IOCRTagHeader.IOCRVLANID	Not checked	—
IOCRTagHeader.IOUserPriority	!= IO CR Priority	RSP-, 0xDB, 0x81, 0x02, 17
IOCRMulticastMACAdd	Case IOCRType == 0x03 or 0x04: Is not multicast adress	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	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
IODataObjectFrameOffset	>= DataLength OR (IODataObjectFrameOffset + (effective SubmoduleDataLength + effective LengthIOPS of corresponding submodule) >= DataLength) OR (IOCRType != "multicast consumer CR" AND (IODataObjectFrameOffset + (effective SubmoduleDataLength + effective LengthIOPS of corresponding submodule) overlap with other IODataObjects or IOCSes))	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

Parameter	Checking rules	Behaviour on match
SubslotNumber	Not in corresponding ExpectedSubmoduleBlockReq OR ((corresponding SubmoduleProperties.Type == 0x0 to 0x1) AND (IOCRTYPE == 0x1)) OR ((corresponding SubmoduleProperties.Type == 0x2) AND (IOCRTYPE == 0x2))	RSP-, 0xDB, 0x81, 0x02, 27
IOCSFrameOffset	>= DataLength OR not unique OR (IOCSFrameOffset + (effective LengthIOCS of corresponding submodule) >= DataLength) OR (IOCSFrameOffset + (effective LengthIOCS of corresponding submodule) overlap with other IODataObjects or IOCSes)	RSP-, 0xDB, 0x81, 0x02, 28

6.2.16.2.4 AlarmCRBlockReq

This field shall be checked according to Table 483.

Table 483 – AlarmCRBlockReq – request check

Parameter	Checking rules	Behaviour 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	> 0xFFF OR != AlarmCRTagHeaderLow. AlarmCRVLANID	RSP-, 0xDB, 0x81, 0x04, 11
AlarmCRTagHeaderHigh. AlarmUserPriority	!= Alarm CR Priority High	RSP-, 0xDB, 0x81, 0x04, 11
AlarmCRTagHeaderLow. AlarmCRVLANID	> 0xFFF OR != AlarmCRTagHeaderHigh. AlarmCRVLANID	RSP-, 0xDB, 0x81, 0x04, 12
AlarmCRTagHeaderLow. AlarmUserPriority	!= Alarm CR Priority Low	RSP-, 0xDB, 0x81, 0x04, 12

6.2.16.2.5 ExpectedSubmoduleBlockReq

This field shall be checked according to Table 484.

Table 484 – ExpectedSubmoduleBlockReq – request check

Parameter	Checking rules	Behaviour 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
ModuleIdentNumber	== 0x00000000	RSP-, 0xDB, 0x81, 0x03, 7
ModuleProperties.reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x03, 8
NumberOfSubmodules	== 0	RSP-, 0xDB, 0x81, 0x03, 9
SubslotNumber	Case ARProperties.PullModuleAlarmAllowed (=0): != 0x0001 to 0x7FFF OR not supported in GSDML OR not unique OR not used in at least one IOCR OR == 0x8000 to 0x8FFF AND API != 0 Case ARProperties.PullModuleAlarmAllowed (=1): != 0x0000 to 0x7FFF OR not supported in GSDML OR not unique OR not used in at least one IOCR OR == 0x8000 to 0x8FFF AND API != 0	RSP-, 0xDB, 0x81, 0x03, 10
SubmoduleProperties.Type	== (NO_IO OR INPUT) AND one output description block follows OR == (OUTPUT OR IO) AND one input description block follows	RSP-, 0xDB, 0x81, 0x03, 12
SubmoduleProperties. SharedInput	If SubmoduleProperties.Type == (OUTPUT DATA) AND SubmoduleProperties.SharedInput == 0x1	RSP-, 0xDB, 0x81, 0x03, 12
SubmoduleProperties. ReduceInputSubmoduleDataLen gth	Not checked	—
SubmoduleProperties. ReduceOutputSubmoduleDataLe ngth	Not checked	—
SubmoduleProperties. DiscardIOXS	Not checked	—
SubmoduleProperties. reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x03, 12

Parameter	Checking rules	Behaviour on match
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	!= 0 to 1 439	RSP-, 0xDB, 0x81, 0x03, 14
LengthIOPS	!= 0x01	RSP-, 0xDB, 0x81, 0x03, 15
LengthIOCS	!= 0x01	RSP-, 0xDB, 0x81, 0x03, 16

6.2.16.2.6 PrmServerBlock

This field shall be checked according to Table 485.

Table 485 – PrmServerBlock – request check

Parameter	Checking rules	Behaviour 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	—
CMInitiatorActivityTimeoutFactor	!= 1 to 1 000	RSP-, 0xDB, 0x81, 0x05, 6
StationNameLength	!= 1 to 240	RSP-, 0xDB, 0x81, 0x05, 7
ParameterServerStationName	Not visible string according to RFC 3490	RSP-, 0xDB, 0x81, 0x05, 8

6.2.16.2.7 MCRBlock

This field shall be checked according to Table 486.

Table 486 – MCRBlockReq – request check

Parameter	Checking rules	Behaviour 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 RFC 3490	RSP-, 0xDB, 0x81, 0x06, 8
Padding	Not checked	—

6.2.16.2.8 ARRPCBlock

This field shall be checked according to Table 487.

Table 487 – ARRPCBlockReq – request check

Parameter	Checking rules	Behaviour 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

6.2.16.3 IODConnectRes**6.2.16.3.1 ArgsLength****6.2.16.3.2 ARRPCBlock**

This field shall be checked according to Table 488.

Table 488 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	Okay, check IODConnectRes-PDU
	RPCOperationNmb	== 1 (Release)	—
	RPCOperationNmb	== 2 (Read)	—

	Parameter	Checking rules	Behaviour on match
	RPCOperationNmb	== 3 (Write)	—
	RPCOperationNmb	== 4 (Control)	—
	RPCOperationNmb	== 5 (Read Implicit)	—
NDRDataResponse	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	—	—	—

6.2.16.3.3 ARBlockRes

This field shall be checked according to Table 489.

Table 489 – ARBlockRes – response check

Parameter	Checking rules	Behaviour 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
ResponderUDPRTPort	== 0x0000 to 0x03FF	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.3.4 IOCRBlockRes

This field shall be checked according to Table 490.

Table 490 – IOCRBlockRes – response check

Parameter	Checking rules	Behaviour 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

6.2.16.3.5 AlarmCRBlockRes

This field shall be checked according to Table 491.

Table 491 – AlarmCRBlockRes – response check

Parameter	Checking rules	Behaviour 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

6.2.16.3.6 ModuleDiffBlock

This field shall be checked according to Table 492.

Table 492 – ModuleDiffBlock – response check

Parameter	Checking rules	Behaviour 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	—
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.QualifiedInfo	Not checked	—
SubmoduleState.Maintenance-Required	Not checked	—
SubmoduleState.Maintenance-Demanded	Not checked	—
SubmoduleState.DiagInfo	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	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

6.2.16.4 IODControlReq

6.2.16.4.1 ArgsLength

This field shall be checked according to Table 493.

Table 493 – ArgsLength check

	Parameter	Checking rules	Behaviour 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	RMPM: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the response	RMPM: Unknown blocks
IODControlReq	—	—	—

6.2.16.4.2 ControlBlockConnect request

This field shall be checked according to Table 494.

Table 494 – ControlBlockConnect – request check

Parameter	Checking rules	Behaviour 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	RMPM: 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

6.2.16.4.3 ControlBlockPlug

This field shall be checked according to Table 495.

Table 495 – ControlBlockPlug – request check

Parameter	Checking rules	Behaviour 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
ARUID	!= ARBlockReq.ARUID	RMPM: AR UID 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

6.2.16.5 IODControlRes**6.2.16.5.1 ArgsLength**

This field shall be checked according to Table 496.

Table 496 – ArgsLength check

	Parameter	Checking rules	Behaviour 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	—	—	—

6.2.16.5.2 ControlBlockConnect

This field shall be checked according to Table 497.

Table 497 – ControlBlockConnect – response check

Parameter	Checking rules	Behaviour 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
ARUUID	!= IODControlReq.ControlBlockConnect. ARUUID	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

6.2.16.5.3 ControlBlockPlug

This field shall be checked according to Table 498.

Table 498 – ControlBlockPlug – response check

Parameter	Checking rules	Behaviour 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
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

6.2.16.6 IOXControlReq**6.2.16.6.1 ArgsLength**

This field shall be checked according to Table 499.

Table 499 – ArgsLength check

	Parameter	Checking rules	Behaviour 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	RMPM / NRPM: ArgsLength invalid
	ArgsLength	General: If one OR more unknown blocks are contained in the request	RMPM / NRPM: Unknown blocks
IOXControlReq	—	—	—

6.2.16.6.2 ControlBlockConnect

This field shall be checked according to Table 500.

Table 500 – ControlBlockConnect – request check

Parameter	Checking rules	Behaviour 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
ARUID	!= ARBlockReq.ARUID	RMPM: AR UID 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

6.2.16.6.3 ControlBlockPlug

This field shall be checked according to Table 501.

Table 501 – ControlBlockPlug – request check

Parameter	Checking rules	Behaviour 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	RMPM: 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

6.2.16.7 IOXControlRes

6.2.16.7.1 ArgsLength

This field shall be checked according to Table 502.

Table 502 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	—
	RPCOperationNmb	== 1 (Release)	—

	Parameter	Checking rules	Behaviour on match
	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	RMPM: ArgsLength invalid
	ArgsLength	General: If one OR more unknown blocks are contained in the response	RMPM: Unknown blocks
IOXControlRes	—	—	—

6.2.16.7.2 ControlBlockConnect

This field shall be checked according to Table 503.

Table 503 – ControlBlockConnect – response check

Parameter	Checking rules	Behaviour 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
ARUUID	!= IOXControlReq.ControlBlockConnect. ARUUID	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

6.2.16.7.3 ControlBlockPlug

This field shall be checked according to Table 504.

Table 504 – ControlBlockPlug – response check

Parameter	Checking rules	Behaviour 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
ARUUID	!= IOXControlReq.ControlBlockPlug. ARUUID	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

6.2.16.7.4 ModuleDiffBlock

This field shall be checked according to Table 492.

6.2.16.8 IODReleaseReq

6.2.16.8.1 ArgsLength

This field shall be checked according to Table 505.

Table 505 – ArgsLength check

	Parameter	Checking rules	Behaviour 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)	—

	Parameter	Checking rules	Behaviour on match
NDRDataRequest	ArgsLength	!= ReleaseBlock.BlockLength + 4	RMPM: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the request	RMPM: Unknown blocks
IODReleaseReq	—	—	—

6.2.16.8.2 ReleaseBlock

This field shall be checked according to Table 506.

Table 506 – ReleaseBlock – request check

Parameter	Checking rules	Behaviour 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
ARUUID	!= ARBlockReq.ARUUID	RMPM: AR UUID 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

6.2.16.9 IODReleaseRes

6.2.16.9.1 ArgsLength

This field shall be checked according to Table 507.

Table 507 – ArgsLength check

	Parameter	Checking rules	Behaviour 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	RMPM: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the response	RMPM: Unknown blocks
IODReleaseRes	—	—	—

6.2.16.9.2 ReleaseBlock

This field shall be checked according to Table 508.

Table 508 – ReleaseBlock – response check

Parameter	Checking rules	Behaviour 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
ARUID	!= IODReleaseReq.ReleaseBlock.ARUID	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

6.2.16.10 IODWriteReq

6.2.16.10.1 ArgsLength

This field shall be checked according to Table 509.

Table 509 – ArgsLength check

	Parameter	Checking rules	Behaviour 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)	RMPM: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the request	RMPM: Unknown blocks
IODWriteReq	—	—	—

6.2.16.10.2 IODWriteReqHeader

This field shall be checked according to Table 510.

Table 510 – IODWriteReqHeader – request check

Parameter	Checking rules	Behaviour 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	—
ARUID	!= ARBlockReq.ARUID	RMPM: AR UID 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 0x8FFF 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
RecordDataLength	!= consistent with ArgsLength	RSP-, 0xDF, 0x81, 0x08, 11
RWPadding	Not checked	—

6.2.16.10.3 RecordDataWrite

This field shall be not checked.

6.2.16.11 IODWriteRes**6.2.16.11.1 ArgsLength**

This field shall be checked according to Table 511.

Table 511 – ArgsLength check

	Parameter	Checking rules	Behaviour 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	—	—	—

6.2.16.11.2 IODWriteResHeader

This field shall be checked according to Table 512.

Table 512 – IODWriteResHeader – response check

Parameter	Checking rules	Behaviour 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	Not checked	—
AdditionalValue1	Not checked	—
AdditionalValue2	Not checked	—
PNIOStatus	Not checked	—

6.2.16.12 IODWriteMultipleReq

6.2.16.12.1 ArgsLength

This field shall be checked according to Table 513.

Table 513 – ArgsLength check

	Parameter	Checking rules	Behaviour 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)	—

	Parameter	Checking rules	Behaviour on match
NDRDataRequest	ArgsLength	$\neq \text{IODWriteReqHeader.BlockLength} + 4 + \text{Sum of all IODWriteReqHeader.BlockLength} + \text{Number of WriteBlocks} * 4$	RMPM: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the request	RMPM: Unknown blocks
IODWriteReq	—	—	—

6.2.16.12.2 IODWriteReqHeader (multiple)

This field shall be checked according to Table 510. The value of the parameter index shall be 0xE040.

6.2.16.12.3 IODWriteReqHeader

This field shall be checked according to Table 510.

6.2.16.12.4 RecordDataWrite

This field shall be not checked.

6.2.16.12.5 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.

6.2.16.13 IODWriteMultipleRes

6.2.16.13.1 ArgsLength

This field shall be checked according to Table 514.

Table 514 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	$\equiv 0$ (Connect)	—
	RPCOperationNmb	$\equiv 1$ (Release)	—
	RPCOperationNmb	$\equiv 2$ (Read)	—
	RPCOperationNmb	$\equiv 3$ (Write)	Okay, check IODWriteMultipleRes-PDU
	RPCOperationNmb	$\equiv 4$ (Control)	—
	RPCOperationNmb	$\equiv 5$ (Read Implicit)	—
NDRData-Response	ArgsLength	$\neq \text{IODWriteResHeader.BlockLength} + 4 + \text{Sum of all IODWriteResHeader.BlockLength} + \text{Number of IODWriteResHeaderBlocks} * 4$	RMPM: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the response	RMPM: Unknown blocks
IODWriteMultiple-Res	—	—	—

6.2.16.13.2 IODWriteResHeader (multiple)

This field shall be checked according to Table 512. The value of the parameter index shall be 0xE040.

6.2.16.13.3 IODWriteResHeader

This field shall be checked according to Table 512.

6.2.16.14 IODReadReq

6.2.16.14.1 ArgsLength

This field shall be checked according to Table 515.

Table 515 – ArgsLength check

	Parameter	Checking rules	Behaviour 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	RMPM: ArgsLength invalid
	—	General: If one OR more unknown blocks are contained in the request	RMPM: Unknown blocks
IODReadReq	—	—	—

6.2.16.14.2 IODReadReqHeader

This field shall be checked according to Table 516.

Table 516 – IODReadReqHeader – request check

Parameter	Checking rules	Behaviour 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	—
ARUID	Case RPCOperationNmb == 2 (Read): != ARBlockReq.ARUID	RMPM: AR UID unknown
ARUID	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 0x8FFF 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
TargetARUID	Case RPCOperationNmb == 2 (Read): TargetARUID != NIL	RSP-, 0xDE, 0x81, 0x08, 12
RWPadding	Not checked	—

6.2.16.14.3 RecordDataReadQuery

This field shall be checked according to Table 517.

Table 517 – RecordDataReadQuery – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0500 OR not supported in GSDML	RMPM: 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	—

6.2.16.15 IODReadRes**6.2.16.15.1 ArgsLength**

This field shall be checked according to Table 518.

Table 518 – ArgsLength check

	Parameter	Checking rules	Behaviour 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	—	—	—

6.2.16.15.2 IODReadResHeader

This field shall be checked according to Table 519.

Withdrawing
IECNORM.COM : Click to view the full PDF of IEC 61158-6-10:2010

Table 519 – IODReadResHeader – response check

Parameter	Checking rules	Behaviour 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	—
ARUID	!= IODReadReqHeader.ARUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
API	!= IODReadReqHeader.API	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SlotNumber	!= IODReadReqHeader.SlotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubslotNumber	!= IODReadReqHeader.SubslotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Index	!= 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	—

6.2.16.15.3 RecordDataRead

This field shall be not checked.

6.3 FAL protocol state machines**6.3.1 Overall structure****6.3.1.1 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.

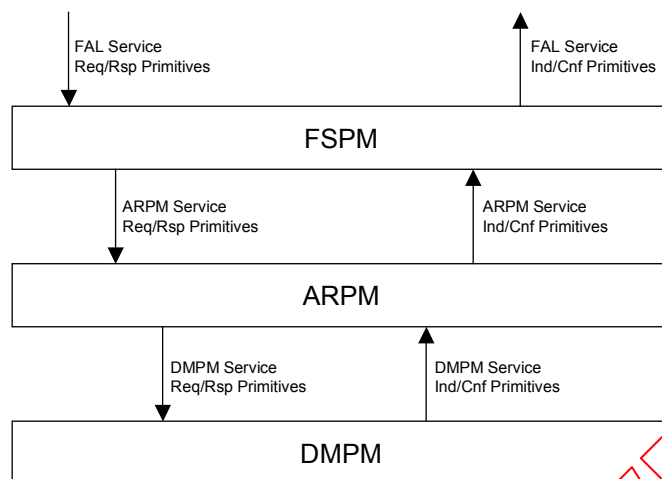


Figure 60 – Relationship among Protocol Machines

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.

6.3.1.2 Fieldbus Service Protocol Machines (FSPM)

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. In addition a basic redundancy control scheme is included that allow to collaborate two AR into a single entity with higher availability.

For this task, the following support shall be offered by an FSPM:

Redundant communication interface (RedCom) to a peer FSPM (only for redundant devices)

Coordination of the Primary/Backup APDU_Status(information)

Issuing switchover Procedures (signal Primary failures in case of an IO-Device)

Collaboration of the set and get data services of the redundant AR

NOTE A switchover will be initiated in case of a communication problem (indicated by CPM via a NoData) if a switchover can be completed before Data hold time expires the system will continue otherwise the consumer will use Substitute Data for further processing.

6.3.1.3 IO controller to IO device (IO AR)

The PPM, CPM, ALPMI, ALPMR, APMR, APMS, CMDEV, CMCTL, RMPM State Machines are responsible for the cyclic, acyclic and alarm data transfer between an IO controller and an IO device.

6.3.1.4 IO supervisor to IO device (Supervisor AR)

The PPM, CPM, ALPMI, ALPMR, APMR, APMS, CMDEV, CMCTL, RMPM State Machines are responsible for the cyclic, acyclic and alarm data transfer between an IO supervisor and an IO device.

NOTE Within this application relationship, the PPM, CPM, APMR, and APMS protocol machines convey their APDUs directly over RPC and not over the LMPM.

6.3.1.5 DLL Mapping Protocol Machines (DMPM)

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.

6.4 AP-Context state machine

There is no AP-Context State Machine defined for this Protocol.

6.5 FAL Service Protocol Machines

6.5.1 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).

6.5.2 FSPMDEV

6.5.2.1 Overview

The FSPMDEV is responsible for the startup of the underlying protocol machines. They shall be started in the following order:

LMPM

LLDP

DHCP/DCP

RMPM (part of address resolution, the RMPM starts all other machines-IP, UDP, RPC, RM (other parts), CMDEV, PPM, CPM ...)

6.5.2.2 Primitive definitions

6.5.2.2.1 Primitives exchanged between FSPMDEV and AP-Context

Table 520 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 520 – Primitives issued by AP-Context (FAL user) to FSPMDEV

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read.rsp(+)	FAL User	AREP Seq Number Add Data 1 Add Data 2 Length Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RSP+
Read.rsp(-)	FAL User	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 Input Data.rsp(+)	FAL User	AREP Seq Number Length Length Data Length IOCS Length IOPS IOCS IOPS Subslot Input Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_INPD_RSP+
Read Input Data.rsp(-)	FAL User	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_INPD_RSP-
Read Output Data.rsp(+)	FAL User	AREP Seq Number Length Length IOCS Length IOPS Length Output Data IOCS IOPS Output Data Substitute Mode Substitute Active Flag Output Substitute Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_OUTPD_RSP+

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read Output Data.rsp(-)	FAL User	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_OUTPD_RS P-
Read Logbook.rsp(+)	FAL User	AREP Seq Number Length Current Local Time Stamp Number Of Log Entries List of Entries Local Time Stamp AR UUID PNIO Status Entry Detail	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_LOG_RSP+
Read Logbook.rsp(-)	FAL User	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_LOG_RSP-
Read Device Diagnosis.rsp(+)	FAL User	AREP Seq Number Length List of Diagnosis Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_DIAG_RSP+
Read Device Diagnosis.rsp(-)	FAL User	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_DIAG_RSP-

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read Expected Identification.rsp(+)	FAL User	AREP Seq Number Length List of Slots	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_EID_RSP+
Read Expected Identification.rsp(-)	FAL User	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_EID_RSP-
Read Real Identification.rsp(+)	FAL User	AREP Seq Number Length Number of Slots List of Slots	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RID_RSP+
Read Real Identification.rsp(-)	FAL User	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_RID_RSP-
Read Identification Difference.rsp(+)	FAL User	AREP Seq Number Length Number of APIs List of APIs	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_IDDIFF_RSP+
Read Identification Difference.rsp(-)	FAL User	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_IDDIFF_RSP-

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read Real Port Data.rsp(+)	FAL User	AREP Seq Number Length Real List of Ports	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RPD_RSP+
Read Real Port Data.rsp(-)	FAL User	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_RPD_RSP-
Read Expected Port Data.rsp(+)	FAL User	AREP Seq Number Length Expected List of Ports	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_EPD_RSP+
Read Expected Port Data.rsp(-)	FAL User	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_EPD_RSP-
Read Adjusted Port Data.rsp(+)	FAL User	AREP Seq Number Length Adjusted List of Ports	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_APD_RSP+
Read Adjusted Port Data.rsp(-)	FAL User	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_APD_RSP-

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read IR Data.rsp(+)	FAL User	AREP Seq Number Length IR Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_IRD_RSP+
Read IR Data.rsp(-)	FAL User	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_IRD_RSP-
Read Real Sync Data.rsp(+)	FAL User	AREP Seq Number Length Sync Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RSYN_RSP+
Read Real Sync Data.rsp(-)	FAL User	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_RSYN_RSP-
Read Expected Sync Data.rsp(+)	FAL User	AREP Seq Number Length Sync Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_ESYN_RSP+
Read Expected Sync Data.rsp(-)	FAL User	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_ESYN_RSP-

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read PDev Data.rsp(+)	FAL User	AREP Seq Number Real List of Ports Expected List of Ports Adjusted List of Ports IR Data Sync Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_PDEV_RSP+
Read PDev Data.rsp(-)	FAL User	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_PDEV_RSP-
Read AR Data.rsp(+)	FAL User	AREP Seq Number Length Number Of ARs List of ARs	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_ARD_RSP+
Read AR Data.rsp(-)	FAL User	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_ARD_RSP-
Read IsoM Data.rsp(+)	FAL User	AREP Seq Number Length IsoM Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_ISOM_RSP+
Read IsoM Data.rsp(-)	FAL User	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_ISOM_RSP-

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Write.rsp(+)	FAL User	AREP Multiple Seq Number Add Data 1 Add Data 2	CM_Write.rsp(+) (AREP, Multiple, Seq Number, AddData1, AddData2)	MAP_WRITE_RSP+
Write.rsp(-)	FAL User	AREP Multiple Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp(-) (AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_RSP-
Write Expected Port Data.rsp(+)	FAL User	AREP Multiple Seq Number	CM_Write.rsp(+) (AREP, Multiple, Seq Number, AddData1, AddData2)	MAP_WRITE_EPD_RSP+
Write Expected Port Data.rsp(-)	FAL User	AREP Multiple Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp(-) (AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_EPD_RSP-
Write Adjusted Port Data.rsp(+)	FAL User	AREP Multiple Seq Number	CM_Write.rsp(+) (AREP, Multiple, Seq Number, AddData1, AddData2)	MAP_WRITE_APD_RSP+
Write Adjusted Port Data.rsp(-)	FAL User	AREP Multiple Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp(-) (AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_APD_RSP-
Write IR Data.rsp(+)	FAL User	AREP Multiple Seq Number	CM_Write.rsp(+) (AREP, Multiple, Seq Number, AddData1, AddData2)	MAP_WRITE_EPD_RSP+

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Write IR Data.rsp(-)	FAL User	AREP Multiple Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp(-)(AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_EPD_RSP-
Set Input.req	FAL User	AREP CREP Slot Number Subslot Number IOPS Subslot Input Data	PPM_Set_Prov_Data.req (CREP,Data)	MAP_SIN_REQ
Set Output IOCS.req	FAL User	AREP CREP Slot Number Subslot Number IOCS	PPM_Set_Prov_Data.req (CREP,Data)	MAP_OIOCS_REQ
Set Input APDU Data Status.req	FAL User	AREP DataValid Flag ARState Flag ProviderState Flag ProblemIndicator Flag	PPM_Set_Prov_Status.req (CREP,D_Status)	MAP_SAS_REQ
Get Input IOCS.req	FAL User	AREP CREP Slot Number Subslot Number	CPM_Get_Cons_Status.req (CREP)	MAP_GINIOCS_REQ
Get Output.req	FAL User	AREP CREP Slot Number Subslot Number	CPM_Get_Cons_Data.req (CREP), CPM_Get_Cons_Status.req (CREP)	CREP=CREP
Alarm Notification.req	FAL User	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

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Alarm Ack.req	FAL User	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier	ALPMR_Alarm_Ack.req(CR EP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, PNIO_Status)	MAP_AA_REQ
Connect.rsp(+)	FAL User	AREP AR Response Block List of IO CR Response Blocks Alarm CR Response Block Module Diff Block	CM_Connect.rsp(+)(AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock)	MAP_CON_RSP+
Connect.rsp(-)	FAL User	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-
Release.rsp(+)	FAL User	AREP Session Key	CM_Release.rsp(+)(AREP, ControlBlock)	MAP_REL_RSP+
Release.rsp(-)	FAL User	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Release.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_REL_RSP-
Abort.req	FAL User	AREP	CM_Abort.req (AREP)	AREP=AREP
End Of Parameter.rsp(+)	FAL User	AREP Session Key Alarm Sequence Number	CM_DControl.rsp(+) (AREP, ControlBlock)	MAP_EOP_RSP+
End Of Parameter.rsp(-)	FAL User	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-
Application ready.req	FAL User	AREP Session Key Alarm Sequence Number Module Diff Block Control Block Properties	CM_CCControl.req(AREP, ControlBlock, ModuleDiffBlock)	MAP_AREADY_REQ

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
local action	—	—	CM_Init.req	None
local action	—	—	TMM_Set_Schedule_Item.req(Port, Sched_list)	None
local action	—	—	TMM_MasterAdd.req (SA, ProtVar, MS_list, MS_status)	None
local action	—	—	TMM_MasterRem.req (SA, ProtVar, MS_list, MS_status)	None
local action	—	—	RCTL_Stop.req	None
local action	—	—	SYN_Delay_Req.req (Port, DA, SA, ADQ_list, ADQ_status)	None
local action	—	—	SYN_Delay_Req.req (Port, DA, SA, ADQ_list, ADQ_status)	None

Table 521 shows the service primitives including their associated parameters issued by the FSPMDEV and received by the AP-Context (FAL user).

Table 521 – Primitives issued by FSPMDEV to AP-Context (FAL user)

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Read.ind(AREP, API, TargetARUUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_IND	Read.ind	FSPM	AREP API Target ARUUUID Slot Number Subslot Number Index Seq Number Length
CM_Read.ind(AREP, API, TargetARUUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_INPD_IND	Read Input Data.ind	FSPM	AREP API Target ARUUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_OUTPD_IND	Read Output Data.ind	FSPM	AREP API Target ARUUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_LOG_IND	Read Logbook.ind	FSPM	AREP Seq Number Length
CM_Read.ind(AREP, API, TargetARUUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_DIAG_IND	Read Device Diagnosis.ind	FSPM	AREP API Target ARUUUID Slot Number Subslot Number Diagnosis Item Seq Number Length
CM_Read.ind(AREP, API, TargetARUUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_EXPECTED_IND	Read Expected Identification.ind	FSPM	AREP Target ARUUUID Slot Number Subslot Number Seq Number Length

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_R ID_IND	Read Real Identification.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I DDIFF_IND	Read Identification Difference.ind	FSPM	AREP Target ARUUID Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_R PD_IND	Read Real Port Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_E PD_IND	Read Expected Port Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_A PD_IND	Read Adjusted Port Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I RD_IND	Read IR Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_R SYNC_IND	Read Real Sync Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_E SYNC_IND	Read Expected Sync Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_P DEV_IND	Read PDev Data.ind	FSPM	AREP Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_A RD_IND	Read AR Data.ind	FSPM	AREP Target ARUUIID Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I SOM_IND	Read IsoM Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I SOM_IND	Read I&M0 Filter Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I SOM_IND	Read I&M0 Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I SOM_IND	Read I&M1 Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I SOM_IND	Read I&M2 Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I SOM_IND	Read I&M3 Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I SOM_IND	Read I&M4 Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_A RD_IND	Read AP Data.ind	FSPM	AREP Seq Number Length
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I SOM_IND	Read Network Component Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Read.ind(AREP, API, TargetARUUIID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_I SOM_IND	Read Real Fiber Optic Data.ind	FSPM	AREP API Target ARUUIID Slot Number Subslot Number Seq Number Length
CM_Write.ind(AREP, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_I ND	Write.ind	FSPM	AREP API Slot Number Subslot Number Index Multiple Prm Flag Seq Number Length Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_ OSUBD_IND	Write Output Substitute Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Substitute Mode Length Output Data Output Substitute Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_ EPD_IND	Write Expected Port Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length Expected List of Ports

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_APD_IND	Write Adjusted Port Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length Adjusted List of Ports
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_IRD_IND	Write IR Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length IR Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_ESYNC_IND	Write Sync Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length Sync Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_ISOIND	Write IsoM Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length IsoM Data

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_I SOM_IND	Write I&M1 Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length IM1 Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_I SOM_IND	Write I&M2 Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length IM2 Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_I SOM_IND	Write I&M3 Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length IM3 Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_I SOM_IND	Write I&M4 Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length IM4 Data

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_I SOM_IND	Write Network Component Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length NC Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_I SOM_IND	Write Fiber Optic Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length FO Data
PPM_Set_Prov_Data.cnf(+) (CREP)	MAP_SIN_CN F	Set Input.cnf(+)	FSPM	AREP
PPM_Set_Prov_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_SIN_CN F	Set Input.cnf(-)	FSPM	AREP
PPM_Set_Prov_Status.cnf(+) (CREP)	MAP_SAS_CN F	Set Input APDU Data Status.cnf(+)	FSPM	AREP
PPM_Set_Prov_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_SAS_CN F	Set Input APDU Data Status.cnf(-)	FSPM	AREP
PPM_Set_Prov_Data.cnf(+) (CREP)	MAP_OIOCS_ CNF	Set Output IOCS.cnf(+)	FSPM	AREP
PPM_Set_Prov_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_OIOCS_ CNF	Set Output IOCS.cnf(-)	FSPM	AREP
CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)	MAP_GOUTD_ CNF+	Get Ouput.cnf(+)	FSPM	AREP IOPS Subslot Output Data New Flag IOCS
CPM_Get_Cons_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_GOUTD_ CNF-	Get Ouput.cnf(-)	FSPM	AREP
CPM_Get_Cons_Status.cnf(+) (CREP, Status, RecvCounter)	MAP_GINIOCS_ CNF+	Get Input IOCS.cnf(+)	FSPM	AREP IOCS
CPM_Get_Cons_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_GINIOCS_ CNF-	Get Input IOCS.cnf(-)	FSPM	AREP
CPM_Set_RedRole.cnf (CREP)	local action	—	—	—

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Init.cnf	local action	—	—	—
CM_Stop.ind	local action	—	—	—
CM_In_Data.ind(CREP)	MAP_INDATA_IND	New Output.ind	FSPM	AREP CREP Slot Number Subslot Number InData Flag
CM_New_Data.ind(CREP, APDU_Status)	MAP_NEWDATA_TA_IND	New Output.ind	FSPM	AREP CREP Slot Number Subslot Number
CM_New_Data.ind(CREP, APDU_Status)	MAP_NEWSTATUS_IND	New Output APDU Data Status.ind	FSPM	AREP CREP Data Flag AR State Flag Provider State Flag Problem Indicator Flag
CPM_NoData.ind(CREP)	MAP_NODATA_IND	New Output.ind	FSPM	AREP CREP Slot Number Subslot Number Watchdog Flag
ALPMI_Alarm_Notification.cnf(+)(CREP)	MAP_AN_CNFP	Alarm notification.cnf(+)	FSPM	AREP
ALPMI_Alarm_Notification.cnf(-)(CREP)	MAP_AN_CNFM	Alarm notification.cnf(-)	FSPM	AREP Status
ALPMI_Alarm_Ack.ind(CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, PNIO_Status)	MAP_AA_IND	Alarm Ack.ind	FSPM	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier
CM_Connect.ind(AREP, ARBlockReq, ListOfIOCRBlockReq, AlarmCRBlockReq, ListOfExpectedSubmoduleBlockReq, PrmServerBlock, ListOfMCRBlockReq)	MAP_CON_IND	Connect.ind	FSPM	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
CM_Release.ind(AREP, ControlBlock)	MAP_REL_IND	Release.ind	FSPM	AREP Session Key
CM_Abort.ind	AREP	Abort.ind	FSPM	AREP
CM_Abort.cnf(AREP)	AREP=AREP	Abort.cnf	FSPM	AREP

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_DControl.ind(AREP, ControlBlock)	MAP_EOP_IND	End Of Parameter.ind	FSPM	AREP Session Key Alarm Sequence Number
CM_CControl.cnf(+)(AREP, ControlBlock)	MAP_AREADY_CNF+	Application Ready.cnf(+)	FSPM	AREP Session Key Alarm Sequence Number
CM_CControl.cnf(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_AREADY_CNF-	Application Ready.cnf(-)	FSPM	AREP Error Decode Error Code 1 Error Code 2
LMPPM_Time_Event.ind(CREP, Cycle)	MAP_SYNCH_IND	SYNCH Event.ind	FSPM	Slot Subslot Global Cycle Counter Phase Status
RCTL_Sync.ind (Port, DA, SA, CS_list, CS_status)	MAP_SYSTI_IND	Sync State Info.ind	FSPM	Real Sync Data Slot Number Subslot Number PTCP Subdomain ID IR Data ID Reserved Interval Begin Reserved Interval End PLL Window Sync Send Factor Send Clock Factor Sync Properties Role Stratum Sync Frame Address PTCP Timeout Factor Sync Error Status Slot Number Subslot Number
DiagnosisEvent.ind(AREP, CREP, API, Diagnosis Data)	MAP_DIAGEV_T_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item
ALPMI_Error.ind(CREP, ERRCLS, ERRCODE)	MAP_DIAGEV_T_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item
DCPUCR_Error.ind (CREP, ERRCLS, ERRCODE)	MAP_DIAGEV_T_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item
DCPMCR_Error.ind (CREP, ERRCLS, ERRCODE)	MAP_DIAGEV_T_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
TMM_MasterAdd.cnf(SA, PortVar, MS_list, MS_status)	local action	—	—	—
TMM_MasterRem.cnf(SA, PortVar, MS_list, MS_status)	local action	—	—	—
LMPM_Schedule_add.cnf (+) (CREP)	local action	—	—	—
LMPM_Schedule_add.cnf (-) (CREP)	local action	—	—	—

6.5.2.2.2 Parameters of FSPMDEV primitives

The parameters used with the primitives exchanged between the FSPMDEV and the AP-Context are described in FAL Service Definition.

6.5.2.3 State machine description

FSPMDEV State Machine has got only one possible state: Run

6.5.2.4 FSPMDEV state table

In general, the FSPMDEV defines no state machine because all services are transferred to the underlying protocol machines.

However, in case of additional multicast communication between IO device the related state machine is defined in Table 522.

Table 522 – FSPMDEV protocol machine for multicast communication

#	Current State	Event /Condition =>Action	Next State
1	run	Set Input.req(Slot Number, Subslot Number, IOPS, Input Data) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected => Data1 := Input Data, IOPS, CREP1 := CREP.IPPM Data2 := Input Data, IOPS CREP2 := CREP.MPPM PPM_Set_Prov_Data.req(CREP1, Data1) PPM_Set_Prov_Data.req(CREP2, Data2)	run
2	run	Set Input.req(Slot Number, Subslot Number, IOPS, Input Data) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero => Data1 := IOPS CREP1 := CREP.IPPM Data2 := Input Data, IOPS CREP2 := CREP.MPPM PPM_Set_Prov_Data.req(CREP1, Data1) PPM_Set_Prov_Data.req(CREP2, Data2)	run
3	run	Get Input IOCS.req(Slot Number, Subslot Number) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected => CREP := CREP.ICPM CPM_Get_Cons_Data.req(CREP)	run
4	run	Get Input IOCS.req(Slot Number, Subslot Number) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero => Get Input IOCS.cnf(-)	run
5	run	Set Output IOCS.req(Slot Number, Subslot Number, IOCS) /submodule with M-Provider functionality && output functionality => CREP := CREP.OPPM PPM_Set_Prov_Data.req(CREP, IOCS)	run
6	run	Set Output IOCS.req(Slot Number, Subslot Number, IOCS) /submodule with M-Provider functionality && NOT (output functionality) => Set Output IOCS.cnf(-)	run

#	Current State	Event /Condition =>Action	Next State
7	run	Get Output.req(Slot Number, Subslot Number) /submodule with M-Provider functionality && NOT(output functionality) => Get Output.req(-)	run
8	run	Get Output.req(Slot Number, Subslot Number) /submodule with M-Provider functionality && output functionality => CREP := CREP.OCPM CPM_Get_Cons_Data.req(CREP)	run
9	run	Set Input APDU Data Status.req(APDU data status flags) /submodule with M-Provider functionality => CREP1 := CREP.IPPM CREP2 := CREP.MPPM D_Status := APDU data status flags PPM_Set_Prov_Status.req(CREP1, D_Status) PPM_Set_Prov_Status.req(CREP2, D_Status)	run
10	run	PPM_Set_Prov_Data.cnf(+) (CREP) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected && CREP == CREP.IPPM => Set Input.cnf(+)	run
11	run	PPM_Set_Prov_Data.cnf(+) (CREP) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero && CREP == CREP.IPPM => Ignore	run
12	run	PPM_Set_Prov_Data.cnf(+) (CREP) /submodule with M-Provider functionality && CREP == CREP.MPPM => Set Input.cnf(+)	run
13	run	PPM_Set_Prov_Data.cnf(+) (CREP) /submodule with M-Provider functionality && CREP == CREP.OPPM => Set Output IOCS.cnf(+)	run

#	Current State	Event /Condition =>Action	Next State
14	run	PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected && CREP == CREP.IPPM => Set Input.cnf(-)	run
15	run	PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero && CREP == CREP.IPPM => ignore	run
16	run	PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Provider functionality && CREP == CREP.MPPM => Set Input.cnf(-)	run
17	run	PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Provider functionality && CREP == CREP.OPPM => Set Output.IOCS.cnf(-)	run
18	run	PPM_Set_Prov_Status.cnf(+) (CREP) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected && CREP == CREP.IPPM => Set Input APDU Data Status.cnf(+)	run
19	run	PPM_Set_Prov_Status.cnf(+) (CREP) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero && CREP == CREP.IPPM => ignore	run
20	run	PPM_Set_Prov_Status.cnf(+) (CREP) /submodule with M-Provider functionality && CREP == CREP.MPPM => Set Input APDU Data Status.cnf(+)	run

#	Current State	Event /Condition =>Action	Next State
21	run	PPM_Set_Prov_Status.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected && CREP == CREP.IPPM => Set Input APDU Data Status.cnf(-)	run
22	run	PPM_Set_Prov_Status.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero && CREP == CREP.IPPM => ignore	run
23	run	PPM_Set_Prov_Status.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Provider functionality && CREP == CREP.MPPM => Set Input APDU Data Status.cnf(-)	run
24	run	CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag) /submodule with M-Provider functionality && CREP == CREP.OCPM => Get Output.cnf(+)(IOPS, Output Data, New Flag, IOCS)	run
25	run	CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag) /submodule with M-Provider functionality && CREP == CREP.ICPM => Get Input IOCS.cnf(IOCS)	run
26	run	CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Provider functionality && CREP == CREP.OCPM => Get Output.cnf(-)	run
27	run	CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Provider functionality && CREP == CREP.ICPM => Get Input IOCS.cnf(-)	run
28	run	CPM_New_Cons_Data.ind(CREP,APDU_Status) /submodule with M-Provider functionality && CREP == CREP.OCPM && no change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run

#	Current State	Event /Condition =>Action	Next State
29	run	CPM_New_Cons_Data.ind(CREP,APDU_Status) /submodule with M-Provider functionality && CREP == CREP.OCPM && change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag) New Output APDU Data Status.ind(APDU status flags)	run
30	run	CPM_New_Cons_Data.ind(CREP,APDU_Status) /submodule with M-Provider functionality && CREP == CREP.ICPM && no change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
31	run	CPM_New_Cons_Data.ind(CREP,APDU_Status) /submodule with M-Provider functionality && CREP == CREP.ICPM && change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag) New Output APDU Data Status.ind(APDU status flags)	run
32	run	CPM_NoData.ind(CREP) /submodule with M-Provider functionality && CREP == CREP.OCPM => Watchdog Flag := WATCHDOG_EXPIRED New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
33	run	CPM_NoData.ind(CREP) /submodule with M-Provider functionality && CREP == CREP.ICPM => Watchdog Flag := WATCHDOG_EXPIRED New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
34	run	CPM_Start.ind(CREP) /submodule with M-Provider functionality => InData Flag := INDATA New Output.ind(Slot Number, Subslot Number, InData Flag) New Output APDU Data Status.ind(AREP, CREP, APDU status flags)	run

#	Current State	Event /Condition =>Action	Next State
35	run	Set Input.req(Slot Number, Subslot Number, IOPS, Input Data) /submodule with M-Consumer functionality && input functionality => Data := Input Data, IOPS CREP := CREP.IPPM PPM_Set_Prov_Data.req(CREP, Data)	run
36	run	Set Input.req(Slot Number, Subslot Number, IOPS, Input Data) /submodule with M-Consumer functionality && NOT (input functionality) => Set Input.cnf(-)	run
37	run	Get Input IOCS.req(Slot Number, Subslot Number) /submodule with M-Consumer functionality && input functionality => CREP := CREP.ICPM CPM_Get_Cons_Data.req(CREP)	run
38	run	Get Input IOCS.req(Slot Number, Subslot Number) /submodule with M-Consumer functionality && NOT (input functionality) => Get Input IOCS.cnf(-)	run
39	run	Set Output IOCS.req(Slot Number, Subslot Number, IOCS) /submodule with M-Consumer functionality => CREP := CREP.OPPM PPM_Set_Prov_Data.req(CREP, IOCS)	run
40	run	Get Output.req(Slot Number, Subslot Number) /submodule with M-Consumer functionality => CREP := CREP.MCPM CPM_Get_Cons_Data.req(CREP)	run
41	run	Set Input APDU Data Status.req(AREP, CREP, APDU data status flags) /submodule with M-Consumer functionality => CREP := CREP.IPPM D_Status := APDU data status flags PPM_Set_Prov_Status.req(CREP, D_Status)	run
42	run	PPM_Set_Prov_Data.cnf(+) (CREP) /submodule with M-Consumer functionality && CREP == CREP.OPPM => Set Output IOCS.cnf(+)	run

#	Current State	Event /Condition =>Action	Next State
43	run	PPM_Set_Prov_Data.cnf(+) (CREP) /submodule with M-Consumer functionality && CREP == CREP.IPPM => Set Input.cnf(+)	run
44	run	PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Consumer functionality && CREP == CREP.OPPM => Set Output IOCS.cnf(-)	run
45	run	PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Consumer functionality && CREP == CREP.IPPM => Set Input.cnf(-)	run
46	run	PPM_Set_Prov_Status.cnf(+) (CREP) /submodule with M-Consumer functionality => Set Input APDU Data Status.req(+)	run
47	run	PPM_Set_Prov_Status.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Consumer functionality => Set Input APDU Data Status.req(-)	run
48	run	CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag) /submodule with M-Consumer functionality && CREP := MCPM => Get Output.cnf(+)(IOPS, Output Data, New Flag)	run
49	run	CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag) /submodule with M-Consumer functionality && CREP := OCPM => ignore	run
50	run	CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag) /submodule with M-Consumer functionality && CREP := ICPM => Get Input IOCS.cnf(IOCS)	run
51	run	CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Consumer functionality && CREP := MCPM => Get Output.cnf(-)	run

#	Current State	Event /Condition =>Action	Next State
52	run	CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Consumer functionality && CREP := OCPM => ignore	run
53	run	CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE) /submodule with M-Consumer functionality && CREP := ICPM => Get Input IOCS.cnf(-)	run
54	run	CPM_New_Cons_Data.ind(CREP,APDU_Status) /submodule with M-Consumer functionality && CREP == CREP.OCPM => ignore	run
55	run	CPM_New_Cons_Data.ind(CREP,APDU_Status) /submodule with M-Consumer functionality && CREP == CREP.ICPM => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
56	run	CPM_New_Cons_Data.ind(CREP,APDU_Status) /submodule with M-Consumer functionality && CREP == CREP.ICPM => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag) New Output APDU Data Status.ind(APDU status flags)	run
57	run	CPM_New_Cons_Data.ind(CREP,APDU_Status) /submodule with M-Consumer functionality && CREP == CREP.MCPM && no change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
58	run	CPM_New_Cons_Data.ind(CREP,APDU_Status) /submodule with M-Consumer functionality && CREP == CREP.MCPM && change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag) New Output APDU Data Status.ind(APDU status flags)	run
59	run	CPM_No_Data.ind(CREP) /submodule with M-Consumer functionality => Watchdog Flag := WATCHDOG_EXPIRED New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run

#	Current State	Event /Condition =>Action	Next State
60	run	CPM_Start.ind(CREP) /submodule with M-Consumer functionality => InData Flag := INDATA New Output.ind(Slot Number, Subslot Number, In Data Flag)	run
61	run	CM_Stop.ind(CREP) /submodule with M-Consumer functionality && CREP == CREP.MCPM => For all submodules of M-Consumer-CR do: M-Provider Communication Stopped.ind(Slot Number, Subslot Number)	run

6.5.2.5 Functions

Table 523 shows the functions defined for FSPMDEV, which are used by service primitives issued by the FAL user.

Table 523 – Functions used by AP-Context (FAL user) to FSPMDEV

Name	Function
MAP_READ_RSP+	map service parameter 1:1 (=)
MAP_READ_RSP-	map service parameter 1:1 (=)
MAP_READ_INPD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Length Data, Length IOCS, Length IOPS, IOCS, IOPS, Subslot Input Data}
MAP_READ_INPD_RSP-	map service parameter 1:1 (=)
MAP_READ_OUTPD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Length IOCS, Length IOPS, Length Output Data, IOCS, IOPS, Output Data, Substitute Mode, Substitute Active Flag, Output Substitute Data}
MAP_READ_OUTPD_RSP-	map service parameter 1:1 (=)
MAP_READ_LOG_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Current Local Time Stamp, Number of Log Entries, [List of Entries (Local Time Stamp, AR UUID, PNIO Status, Entry Detail)]*}
MAP_READ_LOG_RSP-	map service parameter 1:1 (=)
MAP_READ_DIAG_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Diagnosis Item=List of Diagnosis Data
MAP_READ_DIAG_RSP-	map service parameter 1:1 (=)
MAP_READ_EID_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={List of Slots}
MAP_READ_EID_RSP-	map service parameter 1:1 (=)

Name	Function
MAP_READ_RID_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Number of Slots, List of Slots}
MAP_READ_RID_RSP-	map service parameter 1:1 (=)
MAP_READ_IDDIFF_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Number of APIs, List of APIs}
MAP_READ_IDDIFF_RSP-	map service parameter 1:1 (=)
MAP_READ_RPD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Real List of Ports}
MAP_READ_RPD_RSP-	map service parameter 1:1 (=)
MAP_READ_EPD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Expected List of Ports}
MAP_READ_EPD_RSP-	map service parameter 1:1 (=)
MAP_READ_APD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Adjusted List of Ports}
MAP_READ_APD_RSP-	map service parameter 1:1 (=)
MAP_READ_IRD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={{IR Data}}
MAP_READ_IRD_RSP-	map service parameter 1:1 (=)

Name	Function
MAP_READ_RSYNC_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Sync Data}
MAP_READ_RSYNC_RSP-	map service parameter 1:1 (=)
MAP_READ_ESYNC_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Sync Data}
MAP_READ_ESYNC_RSP-	map service parameter 1:1 (=)
MAP_READ_PDEV_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data= {Real Port Data, Expected Port Data, IR Data, Real Sync Data, Expected Sync Data}
MAP_READ_PDEV_RSP-	map service parameter 1:1 (=)
MAP_READ_ARD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Number of ARs, List of ARs}
MAP_READ_ARD_RSP-	map service parameter 1:1 (=)
MAP_READ_ISOM_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={IsoM Data}
MAP_READ_ISOM_RSP-	map service parameter 1:1 (=)
MAP_WRITE_RSP+	map service parameter 1:1 (=)
MAP_WRITE_RSP-	map service parameter 1:1 (=)
MAP_WRITE_EPD_RSP+	map FAL User service parameter 1:1 (=) set AddData1=0 set AddData2=0
MAP_WRITE_EPD_RSP-	map service parameter 1:1 (=)

Name	Function
MAP_WRITE_APD_RSP+	map FAL User service parameter 1:1 (=) set AddData1=0 set AddData2=0
MAP_WRITE_APD_RSP-	map service parameter 1:1 (=)
MAP_WRITE_ISOM_RSP+	map FAL User service parameter 1:1 (=) set AddData1=0 set AddData2=0
MAP_WRITE_ISOM_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
MAP_CON_RSP-	map service parameter 1:1 (=)
MAP_REL_RSP+	AREP=AREP ReleaseBlock.SessionKey=SessionKey
MAP_REL_RSP-	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_EOP_RSP+	AREP=AREP ControlBlockConnect.SessionKey=Session Key^(ControlBlockPlug.SessionKey=Session Key,ControlBlockPlug.AlarmSequenceNumber=Alarm Sequence Number)
MAP_EOP_RSP-	map service parameter 1:1 (=)
MAP_SIN_REQ	CREP=CREP Data={IOPS, Subslot Input Data}
MAP_GINIOCS_REQ	CREP=CREP
MAP_SAS_REQ	CREP=AREP.CREP D_Status={DataValid Flag, AR State Flag, ProviderState Flag, ProblemIndicator Flag}
MAP_OIOCS_REQ	CREP=CREP Data=IOCS
MAP_AN_REQ	map service parameter 1:1 (=)

Table 524 shows the functions defined for FSPMDEV, which are used by services primitives issued by the FSPMDEV.

Table 524 – Function used by FSPMDEV to AP-Context (FAL user)

Name	Function
MAP_READ_IND	if ((Index < =0x7FFF) OR (0xAFF0<=Index<=0xAFFF)) then {AREP=AREP API=API Target ARUUUID = TargetARUUUID Slot Number = SlotNumber Subslot Number =SubslotNumber Index =Index Seq Number =SeqNumber Length=Length} else map to special FAL user Read service primitive
MAP_READ_INPD_IND	if (Index =0x8028) then {AREP=AREP API=API Target ARUUUID = TargetARUUUID Slot Number = SlotNumber Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_OUTPD_IND	if (Index =0x8029) then {AREP=AREP API=API Target ARUUUID = TargetARUUUID Slot Number = SlotNumber Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_LOG_IND	if (Index =0xF830) then {AREP=AREP Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive

Name	Function
MAP_READ_DIAG_IND	if ((Index = 0x800A)OR(Index = 0x800B)OR(Index = 0x800C)OR (Index = 0xC00A)OR(Index = 0xC00B)OR(Index = 0xC00C)OR (Index = 0xE00A)OR(Index = 0xE00B)OR(Index = 0xE00C)OR (Index = 0xF00A)OR(Index = 0xF00B)OR(Index = 0xF00C))then {AREP=AREP if (TargetARUUIID<>NULL) then Target ARUUIID=TargetARUUIID else Target ARUUIID ="IGNORE" if (Index = 0xE00A)OR(Index = 0xE00B)OR(Index = 0xE00C) then API = "IGNORE else API=API if ((Index = 0xF00A)OR(Index = 0xF00B)OR(Index = 0xF00C) OR (Index = 0xE00A)OR(Index = 0xE00B)OR(Index = 0xE00C)) then Slot Number = "IGNORE" else Slot Number =SlotNumber if ((Index = 0xF00A)OR(Index = 0xF00B)OR(Index = 0xF00C) OR (Index = 0xC00A)OR(Index = 0xC00B)OR(Index = 0xC00C)OR (Index = 0xE00A)OR(Index = 0xE00B)OR(Index = 0xE00C)) then Sublot Number = "IGNORE" else Subslot Number =SubslotNumber Seq Number =SeqNumber if ((Index = 0x800A)OR(Index = 0xC00A) OR(Index = 0xE00A)OR(Index = 0xF00A) then Diagnosis Item = CHANNEL if ((Index = 0x800B)OR(Index = 0xC00B) OR(Index = 0xE00B)OR(Index = 0xE00B)OR(Index = 0xF00B) then Diagnosis Item = MANUFACTURER_ERROR_APPEARS if ((Index = 0x800C)OR(Index = 0xC00C) OR(Index = 0xE00C)OR(Index = 0xF00C) then Diagnosis Item = MANUFACTURER_ALL Length=Length} else map to other special FAL user Read service primitive
MAP_READ_EID_IND	if ((Index = 0x8000)OR(Index = 0xC000)OR(Index = 0xE000)) then {AREP=AREP Target ARUUIID = TargetARUUIID if (Index = 0xE000) then Slot Number = "IGNORE" else Slot Number =SlotNumber if ((Index = 0xE000)OR(Index = 0xC000)) then Sublot Number = "IGNORE" else Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive

Name	Function
MAP_READ_RID_IND	if ((Index = 0x8001)OR(Index = 0xC001)OR(Index = 0xE001)OR(Index = 0xF000)) then {AREP=AREP API=API if (Index = 0xF001) then Target ARUUID = "IGNORE" else Target ARUUID = TargetARUUID if ((Index = 0xE001)OR(Index = 0xF001)) then Slot Number = "IGNORE" else Slot Number =SlotNumber if ((Index = 0xE001)OR(Index = 0xC001)OR(Index = 0xF000)) then Sublot Number = "IGNORE" else Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_IDDIFF_IND	if (Index = 0xE002) then {AREP=AREP Target ARUUID = TargetARUUID Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_RPD_IND	if ((Index = 0x802A)OR(Index = 0xC02A)OR(Index = 0xE02A)) then {AREP=AREP Target ARUUID = TargetARUUID if (Index = 0xE02A) then Slot Number = "IGNORE" else Slot Number =SlotNumber if ((Index = 0xE02A)OR(Index = 0xC02A)) then Sublot Number = "IGNORE" else Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_EPD_IND	if ((Index = 0x802B)OR(Index = 0xC02B)OR(Index = 0xE02B)) then {AREP=AREP Target ARUUID = TargetARUUID if (Index = 0xE02B) then Slot Number = "IGNORE" else Slot Number =SlotNumber if ((Index = 0xE02B)OR(Index = 0xC02B)) then Sublot Number = "IGNORE" else Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive

Name	Function
MAP_READ_APD_IND	if ((Index = 0x802F)OR(Index = 0xC02F)OR(Index = 0xE02F)) then {AREP=AREP Target ARUUID = TargetARUUID if (Index = 0xE02F) then Slot Number = "IGNORE" else Slot Number =SlotNumber if ((Index = 0xE02F)OR(Index = 0xC02F)) then Subslot Number = "IGNORE" else Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_IRD_IND	if ((Index = 0x802C)OR(Index = 0xC02C)OR(Index = 0xE02C)) then {AREP=AREP Target ARUUID = TargetARUUID if (Index = 0xE02C) then Slot Number = "IGNORE" else Slot Number =SlotNumber if ((Index = 0xE02C)OR(Index = 0xC02C)) then Subslot Number = "IGNORE" else Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_RSYNC_IND	if ((Index = 0x802D)OR(Index = 0xC02D)OR(Index = 0xE02D)) then {AREP=AREP Target ARUUID = TargetARUUID if (Index = 0xE02D) then Slot Number = "IGNORE" else Slot Number =SlotNumber if ((Index = 0xE02D)OR(Index = 0xC02D)) then Subslot Number = "IGNORE" else Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_ESYNC_IND	if ((Index = 0x802E)OR(Index = 0xC02E)OR(Index = 0xE02E)) then {AREP=AREP Target ARUUID = TargetARUUID if (Index = 0xE02E) then Slot Number = "IGNORE" else Slot Number =SlotNumber if ((Index = 0xE02E)OR(Index = 0xC02E)) then Subslot Number = "IGNORE" else Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive

Name	Function
MAP_READ_PDEV_IND	if (Index =0xF831) then {AREP=AREP Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_ARD_IND	if ((Index =0x8001)OR(Index =0xC001)OR(Index =0xF020)OR(Index =0xF820)) then {AREP=AREP if (Index =0xF820) then API = "IGNORE" else API=API if ((Index =0xF820)OR(Index =0xF020)) then Target ARUUIID = "IGNORE" else Target ARUUIID = TargetARUUIID Slot Number = "IGNORE" Sublot Number = "IGNORE" Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_ISOM_IND	if (Index =0x8030) then {AREP=AREP API=API Target ARUUIID = TargetARUUIID Slot Number = SlotNumber Sublot Number = SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
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

Name	Function
MAP_WRITE_OSUBD_IND	if ((Index =0x801E)) then {AREP=AREP API=API Slot Number = SlotNumber Subslot Number =SubslotNumber Index =Index Multiple = Multiple Seq Number =SeqNumber Length=Length {Substitute Mode, Length Output Data, Output Substitute Data} = Data}
MAP_WRITE_EPD_IND	if ((Index =0x802B)OR(Index =0xC02B)OR(Index =0xE02B)) then {AREP=AREP API=API if (Index =0xE02B) then Slot Number = "IGNORE" else Slot Number = SlotNumber if ((Index =0xE02B)OR(Index =0xC02B)) then Sublot Number = "IGNORE" else Subslot Number =SubslotNumber Index =Index Multiple = Multiple Seq Number =SeqNumber Length=Length {Expected List of Ports} = Data} else map to special FAL user Write service primitive
MAP_WRITE_APD_IND	if ((Index =0x802F)OR(Index =0xC02F)OR(Index =0xE02F)) then {AREP=AREP API=API if (Index =0xE02F) then Slot Number = "IGNORE" else Slot Number = SlotNumber if ((Index =0xE02F)OR(Index =0xC02F)) then Sublot Number = "IGNORE" else Subslot Number =SubslotNumber Index =Index Multiple = Multiple Seq Number =SeqNumber Length=Length {Adjusted List of Ports} = Data} else map to special FAL user Write service primitive